UNIVERSITY OF PORTSMOUTH

DOCTORAL THESIS

Harnessing Emerging Supercomputers for Remote and Interactive Visual Discovery in Astronomy

Timothy DYKES

The thesis is submitted in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy

of the

University of Portsmouth

December, 2018

UNIVERSITY OF PORTSMOUTH

Abstract

Faculty of Creative and Cultural Industries
School of Creative Technologies

Doctor of Philosophy

Harnessing Emerging Supercomputers for Remote and Interactive Visual Discovery in Astronomy

by Timothy DYKES

Scientific visualisation can be an instrumental data analysis tool to effectively extract meaningful information from big data generated via experimentation and observation. Visual analysis can provide a fast and intuitive aid to data comprehension that compliments traditional statistical methods. However, the rate of scientific data production is currently growing well beyond the capabilities of today's processing pipelines, and this situation is expected to be amplified in the coming decades due to the next generation of experimentation and observation facilities and high performance computing systems. The increased size, variety, and complexity of such data presents significant challenges for scientific visualisation, and motivates creation of new and innovative tools to cope with the demands of next-generation research environments in scientific fields such as Astronomy, which is at the forefront of data-intensive disciplines with overwhelming amounts of scientific data both captured by instrumentation and generated by large-scale numerical simulations.

The work presented in this thesis showcases an evolution of an existing tool called Splotch for high performance scientific visualisation, inspired by four core research topics. (1) Physical realism for particle-based volume rendering, in particular considering visual representations for multi-frequency astronomical data and synergies between astronomy and computer graphics algorithms for radiative transport. (2) Optimised utilisation of emerging architectures in modern supercomputing, specifically addressing the growing popularity of accelerator and many-core hardware platforms. (3) Interoperability with modern web-based infrastructures, focusing on a library developed to support integration of interactive high performance computing applications with common web environments for remote and interactive functionalities. (4) Integration within common scientific work-flows demonstrating the role of 3d visualisation in providing appropriate supporting mechanisms for data access and analysis in web portals.

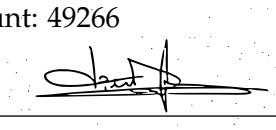
The thesis demonstrates improved quality for astronomical particle rendering, showcasing the application of an enhanced optical model within a novel pipeline for galaxy modelling and visualisation, further laying the groundwork for extended physical realism through coupling with astronomical simulation techniques. Heterogeneous high performance hardware trends are identified, and appropriate optimisation techniques are presented for many-core and graphics processing units,

widening the scope of portability to high performance architectures through kernel-specific and overall efficiency improvements, and outlining methodologies applicable to next-generation architectures. A general approach for integrating high performance computing applications and web-environments is introduced as the foundation for web-based, remote, and interactive functionality in astronomical particle visualisation, and illustrated within the Theoretical Astrophysical Observatory. Future research directions are outlined regarding illumination models, galaxy modelling, performance portability, web visualisation and knowledge transfer beyond astronomy.

Declaration of Authorship

Whilst registered as a candidate for the above degree, I have not been registered for any other research award. The results and conclusions embodied in this thesis are the work of the named candidate and have not been submitted for any other academic award.

Word count: 49266

Signed: 

Date: 30/05/2019

Acknowledgements

First and foremost, I would like to extend my most heartfelt thanks to Mel Krokos (School of Creative Technologies, University of Portsmouth) and Claudio Gheller (formerly ETHZ-CSCS, currently at the Swiss Plasma Center, EPFL, Lausanne, Switzerland) whose advice and support have been of the utmost importance to me, and without either of whom this project would not have been possible.

Throughout this journey I have been fortunate enough to spend a significant period of time collaborating with experts from a wide range of educational and industrial institutions across the world. The time I spent in the company of these distinguished academics and industry veterans was invaluable both academically and personally. My thanks go to Baerbel Koribalski, of the Australia Telescope National Facility, CSIRO, Sydney, Australia; Lucio Mayer, Darren Reed, and Joachim Stadel, along with friends and colleagues at the Institute for Computational Science, University of Zürich, Switzerland; Ugo Varetto, of the Pawsey Supercomputing Centre, Perth, Australia; Adrian Tate and the team in the Cray EMEA Research Lab, Bristol, UK; Amr Hassan, Darren Croton, and friends and colleagues at the Centre for Astrophysics and Supercomputing, Swinburne University of Technology, Melbourne, Australia; Klaus Dolag and Martin Reinecke of the Max Planck Institute for Astrophysics, Garching, Germany; and Marzia Rivi of the Institute of Radioastronomy, INAF, Bologna.

Additional thanks to Malcolm Whitworth (School of Earth and Environmental Sciences), and Gongbo Zhao (Institute for Cosmology and Gravitation), who provided valuable discussion and feedback throughout my time at the University of Portsmouth; and the research degrees staff and coordinators at the University of Portsmouth, including Brett Stevens, who is never short of a word of wisdom (or, at the very least, a sarcastic comment).

Furthermore, I would like to acknowledge the institutions whose funding I have received during the Ph.D. project: the University of Portsmouth and the School of Creative Technologies for paying my bursary and fees for the entirety of my Ph.D.; Santander for awarding me their *Santander Mobility Award* travel funding; the Australia Telescope National Facility, CSIRO, Sydney, Australia, for funding multiple trips and extended stays at the ATNF office in Sydney; the Institute for Computational Science, University of Zürich, Switzerland, for hosting me as a research student for four months; Cray, for hosting me as a research intern within the Cray EMEA Research Lab, Bristol, UK; the Centre for Astrophysics and Supercomputing, Swinburne University of Technology, Melbourne, Australia, for hosting me as a research student for four months.

Parts of this work utilised the GPU Supercomputer for Theoretical Astrophysics Research (gSTAR) national facility at Swinburne University of Technology. The gSTAR facility is funded by Swinburne and the Australian Government's Education Investment Fund. Thanks to Cray for providing further HPC resources used for development, performance, and debugging. Also thanks to UoP and ICG for providing access to Sciama, the High Performance Compute cluster which is supported by the ICG, SEPNet and the University of Portsmouth.

Dedicated to my family, my partner Rose and dog Annie, and my friends, supervisors, and colleagues, without whom I would not have completed this journey.

Dissemination

This thesis is based on the published works:

Dykes, T., C. Gheller, M. Krokos, et al. (2015). “Accelerated 3D visualization of mock galaxy catalogues for the Dark Energy Survey”. In: Astronomical Data Analysis Software and Systems XXV. Astronomical Society of the Pacific.

Dykes, T., C. Gheller, M. Rivi, et al. (2017). “Splotch: porting and optimizing for the Xeon Phi”. In: The International Journal of High Performance Computing Applications 31.6, pp. 550–563.

Dykes, T., A. Hassan, et al. (2018). “Interactive 3D visualization for theoretical virtual observatories”. In: MNRAS 477, pp. 1495–1507.

Reed, D., T. Dykes, et al. (2018). “DIAPHANE: a portable radiation transport library for astrophysical applications”. English. In: Computer Physics Communications 226, pp. 1–9.

Works in preparation:

Dykes, T., U. Varetto, C. Gheller, et al. (2019). "Real-Time Web-Based Remote Interaction with Active HPC Applications". In preparation.

Dykes, T., Gheller, C., Koribalski, B., et al. (2019). "Galaxy Modelling and Visualisation in 3D". In preparation.

Research presentations:

PGR Research Conference: Science Together (08/07/2016)

ATNF 3D Visualization Workshop (16/11/2016)

CSIRO Computational and Data Intensive Science (20/07/2017)

ADACS Data Intensive Astronomy (08/08/2017)

List of Abbreviations

2D	**2** **D**imensional
3D	**3** **D**imensional
ALP	**A**pplication **L**ayer **P**rotocol
ASVO	**A**ll-**S**ky **V**irtual **O**bservatory
ATCA	**A**ustralia **T**elescope **C**ompact **A**rray
AU	**A**stronomical **U**nit
AWS	**A**mazon **W**eb **S**ervices
CAS	**C**enter for **A**strophysics and **S**upercomputing
CIVR	**C**onstrained **I**nverse **V**olume **R**endering
DP	**D**ouble **P**recision
DRN	**D**irect **R**emote **N**ative
DRW	**D**irect **R**emote **W**eb
EM	**E**lectro**m**agnetic
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FLD	**F**lux **L**imited **D**iffusion
FLOP	**F**loating **P**oint **O**peration
FPS	**F**rames **P**er **S**econd
GALEX	**Gal**axy Evolution **Ex**plorer
GAVO	**G**erman **A**strophysical **V**irtual **O**bservatory
GB	**G**iga**B**yte
GPU	**G**raphics **P**rocessing **U**nit
GPGPU	**G**eneral **P**urpose **G**raphics **P**rocessing **U**nit
H I	Atomic Hydrogen
HID	**H**uman **I**nterface **D**evice
HPC	**H**igh **P**erformance **C**omputing
HPSV	**H**igh **P**erformance **S**cientific **V**isualisation
IaaS	**I**nfrastructure **a**s **a** **S**ervice
IRN	**I**ndirect **R**emote **N**ative
IRW	**I**ndirect **R**emote **W**eb
I/O	**I**nput/**O**utput
IR	**I**nfra**r**ed
IRAC	**I**nfrared **A**rray **C**amera
ISA	**I**nstruction **S**et **A**rchitecture
Intel IMCI	Intel **I**nitial **M**any-**C**ore **I**nstructions
IVOA	**I**nternational **V**irtual **O**bservatory **A**lliance
JSON	**J**ava**S**cript **O**bject **N**otation
KNC	**K**nights **C**orner
KNL	**K**nights **L**anding
KPC	**K**ilo-**p**arse**c**
Intel LEO	Intel **L**anguage **E**xtensions for **O**ffload
MB	**M**ega**B**yte
MDL	**M**achine **D**ependent **L**ayer
MPC	**M**ega-**p**arse**c**

MPI	**M**essage **P**assing **I**nterface
MIC	**M**any **I**ntegrated **C**ore
NGC	**N**ew **G**eneral **C**atalogue (of Nebulae and Clusters of Stars)
OMP	**O**pen**MP**
PA	**P**osition **A**ngle
PaaS	**P**latform **a**s **a** **S**ervice
PB	**P**eta**B**yte
PBS	**P**ortable **B**atch **S**ystem
RGB	**R**ed, **G**reen, and **B**lue
RISC	**R**educed-**I**nstruction-**S**et-**C**omputer
RPC	**R**emote **P**rocedure **C**all
SaaS	**S**oftware **a**s **a** **S**ervice
SINGG	**S**urvey for **I**onization in **N**eutral **G**as **G**alaxies
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SIMT	**S**ingle **I**nstruction **M**ultiple **T**hread
SP	**S**ingle **P**recision
SPH	**S**moothed **P**article **H**ydrodynamics
SPMD	**S**ingle **P**rogram **M**ultiple **D**ata
SSH	**S**ecure **Sh**ell
STL	**S**tandard **T**emplate **L**ibrary
TB	**T**era**B**yte
TAO	**T**heoretical **A**strophysical **O**bservatory
TLB	**T**ranslation **L**ook-aside **B**uffer
UV	**U**ltra**v**iolet
VPU	**V**ector **P**rocessor **U**nit
VNC	**V**irtual **N**etwork **C**omputing

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The core motivation of this thesis is to better equip scientists for knowledge discovery in the current data-intensive era of science. An effective means of knowledge discovery is scientific visualisation; the presented work addresses big data challenges in the context of scientific visualisation, specifically focusing on large particle-based datasets such as those produced by N-body and hydrodynamical simulations. This chapter provides a brief introduction to the key concepts and research questions of the thesis in Sections 1.1 to 1.4, outlines the objectives of each research chapter and how they relate to the research questions in Section 1.5, and summarises the thesis contents in Section 1.6.

1.1 Data-Intensive Science and Big Scientific Data

1.1.1 The Rise of Big Scientific Data

Data-intensive scientific discovery is heralded as the fourth paradigm of science (Hey, Tansley, and Tolle, 2009). In many scientific domains, data production through experimentation, observation, and simulation is increasing at an exponential rate, and has been for a significant period of time. Szalay and Gray (2006) pointed out over a decade ago that scientific data of the time was doubling every year and predicted that *scientists in 2020 will continue to work in an exponential world*. We are now nearing 2020, and this prediction is looking ever more likely to hold true; big scientific data is still increasing at an exponential rate, leading the world into a data-rich era of scientific discovery. The challenges of this era, namely effective data storage, movement, processing, analysis, and access, are being faced in many domains; for example, Astronomy (Norris, 2011; Jones, Wagstaff, et al., 2012; Zhang and Zhao, 2015), Medicine (McCue and McCoy, 2017; Kim and Groeneveld, 2017), Climate Sciences (Fiore et al., 2016; Karpatne and Kumar, 2017), Material Sciences (Agrawal and Choudhary, 2016), and other fields inside and outside of science (Szalay, 2011; Demchenko et al., 2013; Jagadish et al., 2014).

Much of the data generated in pursuit of scientific discovery today is described as big, however there is not an exact, quantitative, general definition of what constitutes big data, with the term used relatively across many domains of science and industry. A qualitative definition that is commonly used is a collection of terms

known as *V's*. Early definitions included three *V's*, Volume, Variety, and Velocity (Laney, 2001), and later definitions have extended these; as an example, an extensive systematic literature review of general big data challenges by Sivarajah et al. (2017) identify seven such *V's* to describe the challenges presented by the inherent characteristics of big data. In the context of this work, big data can be sufficiently described by four *V's*:

Volume: The volume of data is large, and such size is beyond the capabilities of typical computing capabilities.

Variety: The variety of data describes the rich diversity in the nature of big data, containing many different types of data field, with a wide range of complex data structures in-memory, and file formats on disk.

Velocity: The high rate at which data is produced or captured.

Value: The utility of big data and the knowledge that can be extracted from it.

A particularly important requirement imposed by big data is the need for *parallel and distributed computing* approaches to address these four *V's*. For a thorough introduction to parallel and distributed computing as a field, the reader is referred to, for example, Tanenbaum and Steen (2007) and Coulouris et al. (2011). A core focus of this thesis will be the use of such approaches to address big data challenges in the context of scientific visualisation for astronomy.

1.1.2 Big Data in Astronomy

Astronomy was amongst the first of the scientific fields to face these challenges caused by an influx of big data (Feigelson and Babu, 2012). Techniques for image collection, storage, and analysis are becoming continually more data-intensive in an effort to observe as much of the sky as possible, in an ever-increasing amount of detail, to answer some of the largest questions about the origin of the Universe and the phenomena within it. Starting from early catalogues captured by hand, such as the New General Catalogue of Nebulae and Clusters of Stars in the 19th century (Dreyer, 1888), astronomers have progressed to all-sky surveys, searching and cataloguing the sky via high precision telescopic instrumentation that produce massive amounts of raw data at a rate that has been roughly doubling every two years (Ferguson et al., 2009). Astronomy is leading the way in data intensive science, pushing the boundaries of big scientific data through both ground and space based experiments and facilities, as can been seen in the examples summarised in Table 1.1. The most extreme of these examples is the upcoming Square Kilometre Array (SKA), which will be the worlds largest radio telescope when complete (expected in 2024), and is expected to produce raw data on a scale of petabytes per second requiring software pipelines and computing facilities far beyond our current capabilities (Norris, 2011).

Facility	**Expected Full Operation**	**Data Challenges**	**References**
Australian Square Kilometre Array Pathfinder (ASKAP)	2019	• ∼2.8 GB/s data ingest • ∼240 TB/day data ingest • ∼5 PB/year data product	Bastholm et al. (2018)
The Euclid Mission	2020	• ∼100 GB/day data ingest • ∼20 TB in 6 year lifecycle • Total of tens of PB, upper limit 100, post-data analysis	(Dubath et al., 2017)
The James Webb Space Telescope (JWST)	2021	• ∼232 GB/day data ingest	Gardner et al. (2006)
The Large Synaptic Survey Telescope (LSST)	2022	• ∼15 TB/day data ingest • ∼500 PB imaging data, ∼50 PB of catalogue databases in 10 year lifecycle	Jurić et al. (2017)
The Square Kilometre Array (SKA)	2024	• ∼5-10 GB/s data ingest per dish, total multiple PB/s ingest • ∼5-10 PB/day imaging data	Faulkner et al. (2010)
The Wide-Field Infrared Survey Telescope (WFIRST)	mid 2020s	• ∼36 MB/s data rate • ∼1.3 TB/day	Faulkner et al. (2010)

TABLE 1.1: The data challenges presented by upcoming astronomical ground and space based telescope facilities.

In addition to the large data rates produced by experimental and observational facilities, computational simulations have been growing in size and producing increasingly larger data outputs. Computational simulations are important for the understanding of theory, providing a means of experimentation for astronomers that is not possible otherwise. A successful approach is that of N-body simulations, which follow the evolution of a dynamical system of particles to experimentally reproduce physical phenomena on many scales, from star and planet formation to the large scale structure of the Universe. Such simulations also provide an integral support for observational missions and facilities, providing a means to test and validate processing pipelines in advance. As an example, Euclid is a European Space Agency funded astronomy and astrophysics space mission aiming to understand the nature of dark energy and the accelerating expansion of the Universe. The cosmological N-body simulation of Potter, Stadel, and Teyssier (2017), the worlds largest dark-matter particle simulation at the time, provides the basis of the Euclid Flagship mock galaxy catalogue; this synthetic replication of the expected catalogue of Euclid survey data will be used for developing the data and science pipelines and assessing the performance of the mission through Science Performance Verification (Euclid Consortium, 2017). Further examples can be seen for other observational surveys (Connolly et al., 2014; MacCrann et al., 2018). Simulations such as this typically require large scale computing resources in order to create the resolution necessary to study physical phenomena on an astronomical scale. As the processing capacity of modern high performance computing resources expands, simulations can be run at higher resolution and in larger computational domains, in turn providing more opportunities for data-intensive scientific discovery. To illustrate this point, Figure 1.1 (courtesy of Volker Springel, Max-Planck-Institute for Astrophysics, Garching, Germany) shows the increasing size of N-body simulations in recent years, in terms of number of simulated particles. This growth is similarly reflected in an equivalent Figure 1.2 for SPH simulations (courtesy of the Illustris Collaboration[1]). Figure 1.1 demonstrates an approximate increase of two orders of magnitude of simulation particles each decade ($\sim 10^6$ in 1990, $\sim 10^8$ in 2000, $\sim 10^{10}$ in 2010); the state of the art N-body simulation of Potter, Stadel, and Teyssier (2017), almost a decade later, exploits 10^{12} particles. As such it is clear the ongoing growth in data rates for scientific simulation are continuing, and likely to continue in future years.

1.1.3 The Challenges of Big Astronomical Data

As scientific data in Astronomy grows, so too does the need for effective tools and techniques to help the scientist to extract *Value*. Such data presents significant challenges to existing analysis tools, relating to:

[1] http://www.illustris-project.org/about/

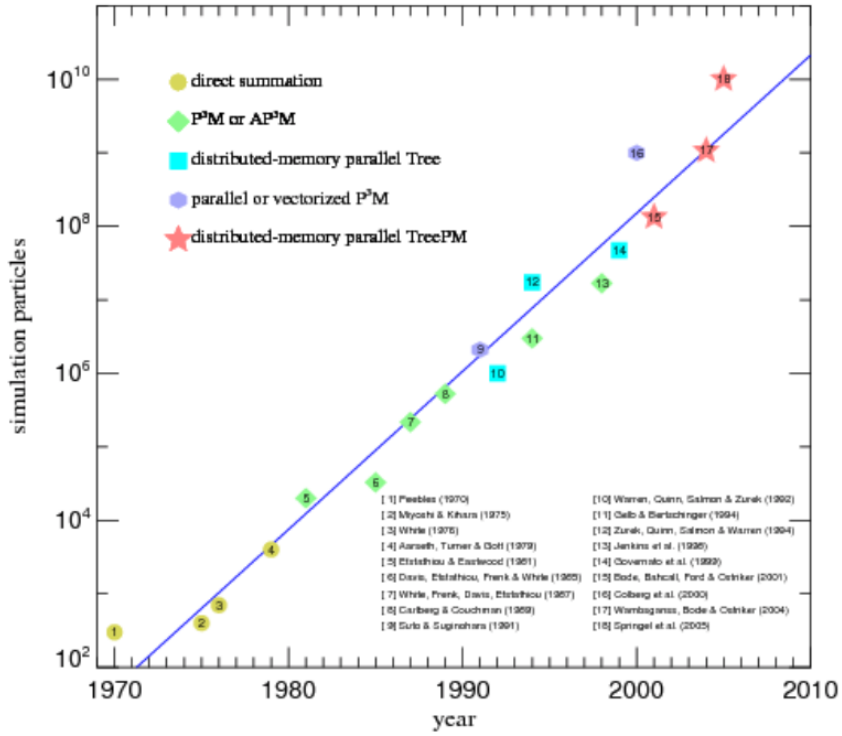


FIGURE 1.1: The consistent increase in N-body simulation size with time, showing approximately two orders of magnitude per decade. Figure courtesy of Volker Springel, Max-Planck-Institute for Astrophysics, Garching, Germany.

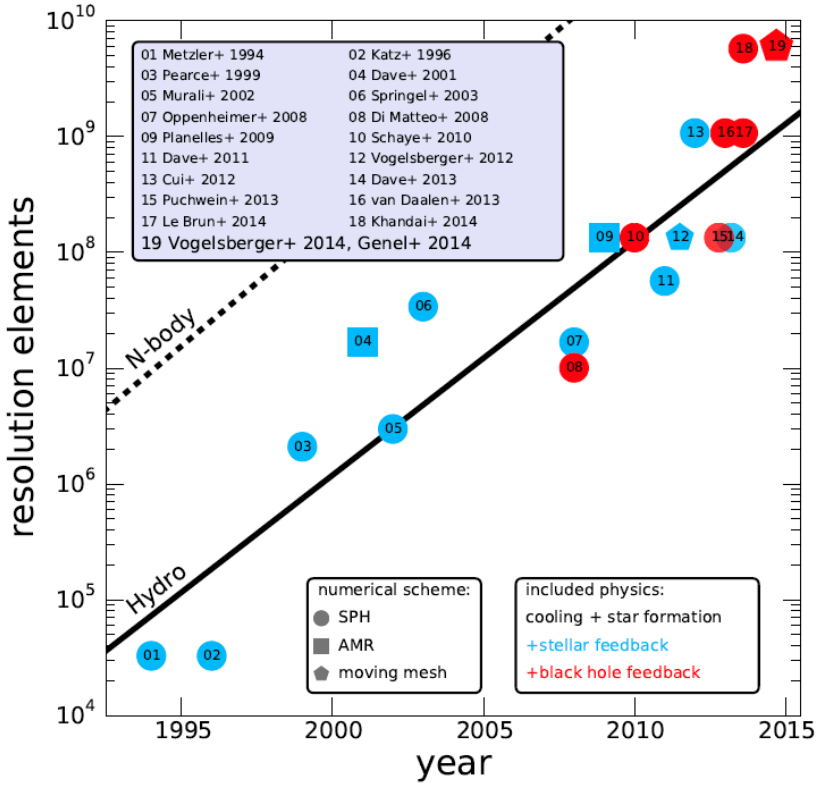


FIGURE 1.2: The consistent increase in SPH simulation size with time, also showing approximately two orders of magnitude per decade, albeit from a smaller starting point than the N-Body simulations of Figure 1.1, due to the more complex computation required for SPH. Figure courtesy of the Illustris Collaboration.

- **Data Processing**:

  There are many different analysis methods for scientific data, from statistical methods to full 3D scientific visualisation. Typically, such analysis tools require the full dataset in memory for processing. However, the *Volume* of big datasets means that loading the whole dataset at one time requires parallel and distributed computing systems with sufficient memory. Many long-standing analysis tools for extracting *Value* are not designed to exploit parallel and distributed computing systems, and those that do typically can not scale sufficiently to exploit the concurrency necessary for the next generation of high performance computing systems. The inability of traditional tools and infrastructures to cope with such data is a significant challenge for astronomy (e.g. Zhang and Zhao (2015), Zhou and Huang (2017)) and science in general (Demchenko et al., 2013).

- **Data Movement & Storage**:

  The *Volume* of scientific data can become infeasible to move in a realistic amount of time (see Table 1.2), especially for ad-hoc analysis during an exploratory scientific workflow. Furthermore, the *Velocity* at which it arrives means it may not be feasible to locally store for an extended period of time. The influx of observational data in large scale astronomical projects is already at such a rate that it cannot be transported or stored in raw form, necessitating in-situ processing and reduction for new generations of instrumentation (Norris, 2011; Jones, Wagstaff, et al., 2012). As a result, users must move the processing to the data (also known as *moving the question to the data* (Hey, Tansley, and Tolle, 2009)). Big scientific datasets are typically stored close to the computing systems with sufficient processing capacity for analysis, for example, on large parallel file systems of supercomputing centres. Such data requires tools that can exploit in-situ and remote approaches for effective data analysis.

- **Data Access & Sharing**:

  The difficulties in storing and moving big research data also presents challenges in allowing researchers to access and share datasets to extract maximal *Value* from the data. The *Variety* of such data can also inhibit sharing, the range of data types, structures, and file types in astronomy (e.g. Brunner et al. (2002) and Hassan and Fluke (2011)) requires not only appropriate tools for such heterogeneous data but also appropriate platforms to distribute comprehensive data to researchers. One approach astronomers are taking to address these challenges revolves around the introduction of research data portals and the Virtual Observatory concept (Djorgovski and Williams, 2005). Analysis tools for large datasets must integrate with such portals, to support researchers in accessing and analysing remote datasets.

File Size	10 Gigabit/s	60 Megabit/s
1 Gigabyte	∼1 second	∼2 minutes
1 Terabyte	∼13 minutes	∼37 hours
1 Petabyte	∼9 days	∼4 years

TABLE 1.2: The increasing cost of data movement as big scientific data grows, for a typical high speed local area network connection and a typical high speed wide area network connection.

1.2 Scientific Visualisation

1.2.1 A Brief Introduction

Visualisation is the transformation of abstract data into a visual representation, with the aim of facilitating human comprehension. Data visualisation has a long history, as illustrated in the review of Friendly (2006), however the focus of this thesis is more specifically on *scientific* visualisation, a shortening of the phrase *'visualisation in scientific computing'* coined by McCormick, DeFanti, and Brown (1987) in the early beginnings of the field. Scientific visualisation concentrates on transforming scientific data, typically 3 dimensions or more of spatial or spatio-temporal data, into imagery to help understanding or analysis; other types of data visualisation, e.g. representing relationships in abstract data, are typically categorized separately as *information visualisation* (Friendly, 2009) and are out of scope for this thesis.

Scientific visualisation today plays a central and indispensable role in contemporary science (Bethel, 2012). As an interdisciplinary field, scientific visualisation interconnects natural sciences with computer sciences such as computer graphics and distributed and parallel computing; the full breadth of research in this field is out of the scope of this thesis but may be seen in various field survey books over the years (Brodlie et al., 1992; Johnson and Hansen, 2004; Bethel, Childs, and Hansen, 2012). Scientific visualisation has been applied in many different contexts in an effort to support the knowledge discovery process, as an example see the review by Lipsa et al. (2012) on visualisation techniques for physical sciences. In astronomy, scientific visualisation has been used extensively in research to analyse data and help draw conclusions, and also to convey concepts via talks and scientific papers. This includes journal special issues and scientific workshops focused on visualisation techniques (e.g. Kent (2017), the AstroVis workshop[2]), and usage in education settings, public outreach, and entertainment (Isik-Ercan, Kim, and Nowak, 2010; Goodman, Udomprasert, et al., 2011; Thorne, 2014). Astronomical visualisation is widely accepted as an established means of analysing scientific data through *exploratory* visualisation, and communicating scientific concepts and conclusions through *explanatory* visualisation (Goodman, Borkin, and Robitaille, 2018).

[2] https://conference.ipac.caltech.edu/astroviz2018/

1.2.2 High Performance Scientific Visualisation

The utility and benefits of tools and techniques for scientific visualisation are clearly established; however, they are also exposed to the challenges of big data. A typical definition of big data for scientific visualisation is that used by Childs (2012), *data that is too large to be processed, in its entirety, all at one time, because it exceeds the available memory*. To elaborate on this definition, which represents the *volume* characteristic of the general definition of big data in Section 1.1, the data size (plus intermediate visualisation data) is larger than the memory available in the computational resources used for visualisation, and as such the entire dataset cannot be loaded into memory at one time for processing. There are significant bodies of research into how best to visualise such data, many of which also address other *V's* of big data. Such research is generally categorised under the domain of *high performance scientific visualisation* (HPSV), a thorough overview of which is provided by Bethel, Childs, and Hansen (2012), and includes a variety of research topics:

- **Parallel and distributed computing for visualisation:**

  The *volume* of big scientific data can make *data processing* slow, or even infeasible. The use of parallel and distributed computing resources can enable visualisation of data larger than the typical memory of a user machine, however requires research into topics such as parallel visualisation pipelines, distributed visualisation algorithms, and data distribution and image compositing techniques.

- **Data Input & Output:**

  Data I/O can become a slow or entirely infeasible process with a large *volume*. Visualisation tools must exploit appropriate methods of *data movement & storage*, to enable efficient loading of data into memory and writing of outputs to disk. This includes techniques such as parallel I/O, data formats for big datasets, data compression, and out-of-core data streaming.

- **Data representation:**

  The representation of data in memory has a strong effect on the efficiency of data *processing*, as well as how it is then *moved, stored, accessed,* and *shared*. Visualisation tools must be capable of parsing the *variety* of complex data structures, as well as efficiently accessing or converting data elements for visualisation. This includes research into topics such as appropriate visualisation techniques for a variety of data structures, efficient data representations for specific visualisation techniques, and multi-resolution approaches to data representation.

- **Visual representation:**

  The final visual representation of data is strongly related to the *value* that can be extracted through visualisation. However, the large *variety* of data requires

a variety of appropriate representation methods, and the *volume* of such data requires parallel approaches for *data processing*. This includes topics such as visual representations for different data types and sources, parallel rendering algorithms, transfer functions, data filtering, levels of detail, and data transformations.

- **Visualisation for current and future architectures:**

  Efficient use of the underlying hardware is an important factor for visualisation, which typically has strong performance requirements for *data processing* (e.g. for real-time user interaction). With such a large data *volume*, visualisation software must be able to efficiently exploit the available hardware of the platform on which it is running. Modern hardware is constantly evolving, as such this includes research into topics such as hardware optimisation (e.g. optimised rendering algorithms for Graphics Processing Units (GPUs)) and approaches for visualisation using other emerging technologies.

- **Remote and in-situ methods:**

  The *volume* of scientific data, as well as how it is *stored* and *accessed*, presents difficulties for *data movement*. Large data is slow to transport, and often simply cannot be transported to a user machine due to the storage capacity required. Remote visualisation approaches move analysis to the data, allowing the use of large scale remote computing resources for visualisation. In-situ and in-transit methods further allow data to be visualised while actively being generated, or during transit from simulation to storage, further reducing the need for data movement.

- **User display and interaction:**

  This topic encompasses the myriad ways in which the user can be presented with visualisation results. Including research into user interfaces for remote or local visualisation, image compression and streaming, tiled rendering for large displays, multi-tile displays, and integration with user workflows and tools.

1.3 Research Questions

The aim of this work is to, in part, address the challenges of big scientific data for astronomy through a modern high performance scientific visualisation approach. The thesis will focus on four specific research questions, derived from the challenges as outlined in Section 1.1.3, that together span the field of high performance visualisation for astronomy. These questions are listed below, following the order in which they will be addressed in the thesis (discussed further in Section 1.5). To illustrate how the research questions relate to the field of study, each is followed by a list of the most relevant high performance visualisation topics from the list in Section 1.2.2.

Q.1 How can domain expertise in astronomy be used to physically motivate volume rendering of astronomical data?

This question focuses on *visual representation* and *data representation*.

Q.2 How can high performance visualisation cope with modern heterogeneous high performance computing systems?

This question focuses on *parallel and distributed computing for visualisation*, and *visualisation for current and future architectures*.

Q.3 How can modern astronomical tools exploit remote and interactive high performance visualisation on the web?

This question focuses on *parallel and distributed computing for visualisation*, *data input & output* and *remote and in-situ methods*.

Q.4 How can remote high performance visualisation facilitate access and analysis of large astronomical datasets on the web?

This question focuses on *data input & output*, *remote and in-situ methods*, and *user display and interaction*.

Figure 1.3 illustrates the mapping between these questions and the three core challenges of big astronomical data. The following subsections, 1.3.1 to 1.4, introduce and contextualise the research questions and software context of the thesis. The questions are then employed to define a series of specific research objectives in Section 1.5, each of which forms the subject of a dedicated thesis chapter.

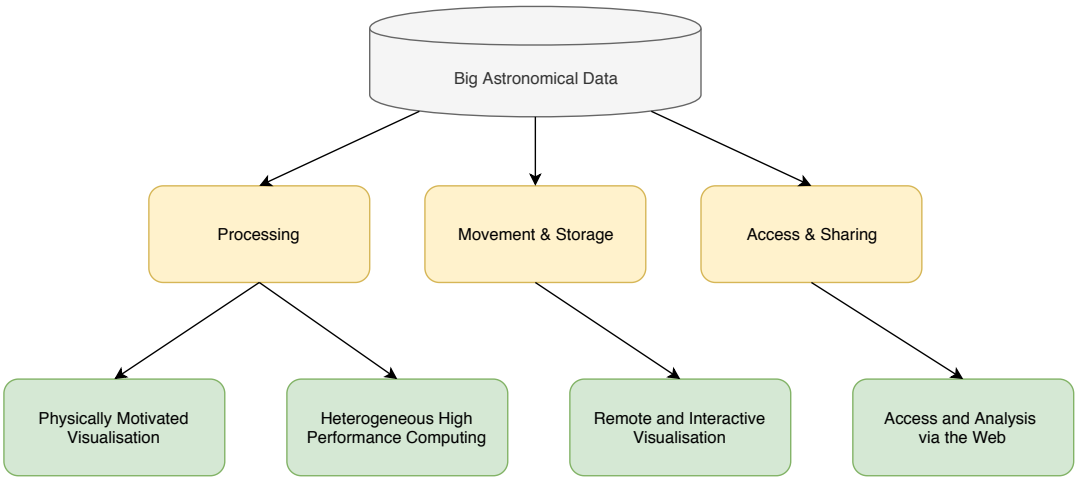


FIGURE 1.3: Illustration of how the general challenges of big scientific data map to the specific research questions of this thesis.

1.3.1 Physically Motivated Particle Visualisation

Visualisation can be used to analyse many different types of data, with a wide variety of visual representations to support different scenarios. Vohl, Barnes, et al. (2016) distinguish such scenarios as: *qualitative visualisation*, supporting intuitive

comprehension of data through shape and colours in images; *quantitative visualisation*, that ties numerical information to the imagery through approaches such as labelled colour bars or filters on data fields; *comparative visualisation*, allowing one to visually compare and contrast multiple datasets; and *collaborative visualisation* where multiple researchers come together to collectively perform visual analysis of data. Visualisation can also be distinguished as *exploratory* and *explanatory*, as previously mentioned in Section 1.2.1. Such classes are not mutually exclusive, e.g. a collaborative visualisation session comparing a variety of quantitative and/or qualitative images, however each class of visualisation places different requirements on visualisation tools. For example, the features required for collaborative visualisation may focus more on display environments and interaction techniques, whilst for quantitative visualisation there is higher emphasis on data management, numerical algorithms, and scientific meta-data.

In terms of visual representation, there is a wide range of options in computer graphics, from simple block coloured 2 dimensional (2D) lines and shapes to highly realistic 3 dimensional (3D) renderings of complex scenes. When describing and categorising such representations, the concept of *realism* is often employed; however, the meaning of realism must be clearly defined for such a description to be valid. Ferwerda (2003) introduced three categories of realism:

- Physical realism

  The image provides the same *visual stimulation* as the scene. This requires: a physically accurate description of the scene, including shapes, material and illumination properties; a physically accurate simulation of light transport to model the emission of light and interactions between source and observer to produce the correct observed radiance; and finally a display capable of accurately representing such radiance to the viewer. Achieving this type of realism is typically computationally expensive, and in many cases not possible to reproduce fully (for example, computer displays almost universally restrict the final step of this process), however can provide interesting benefits such as enabling quantitative image-based analysis for simulated images.

- Photorealism

  The image provides the same *visual response* as the scene. This describes the inability of the viewer to distinguish between a photograph and the image. A more objective definition here is that the image be *photo-metrically* realistic, producing the same visual response in the eye of the viewer as the image. A typical approach to achieving photorealism is to focus on improved physical realism; a natural assumption to make is that physically realistic images are likely to also be photo-metrically realistic images. However, such approaches are typically very computationally expensive; the photo-metric definition of realism allows algorithms that account for imperfection of vision.

Such perceptually-motivated rendering techniques may be less physically accurate, but provide a suitable photometric response for the viewer in a more efficient manner.

- Functional realism

  The image provides the same *visual information* as the scene. This describes the fidelity of the information about the scene that can be comprehended by the viewer. Functionally-motivated images may not have any semblance of photorealism, but provide a realistic understanding of the content of the image. Technical illustrations such as those found in manuals are typically stylised depictions intended to to instruct the viewer in completing a task, as such they are functionally realistic, without need for photo- or physical- realism.

Each of these categories of realism has relevance for scientific visualisation. For example, functional realism is often a crucial component of explanatory visualisation, where the focus is on supporting comprehension of a complex concept. In such cases a photo-realistic representation may be infeasible to generate, or may do little to convey the information supporting comprehension of an underlying theoretical concept. Similarly, qualitative visualisation methods often do not rely on physical realism, and depend solely on intuitive comprehension. Conversely, physical realism can also be an important component of scientific visualisation, both in underpinning photo-realistic approaches and in generating images that can be quantitatively analysed or enabling comparative visualisation with experimental observation. In the scope of this work, the term *physically motivated visualisation* refers to approaches that seek to illustrate physical realism, either through physically realistic rendering algorithms or through physically realistic definitions of functional visual representations.

The nature of astronomical data can be broadly categorised according to Brunner et al. (2002) and summarised as in Hassan and Fluke (2011):

- **Imaging Data:** two-dimensional images representing a specific electromagnetic range at a specific instant of time.
- **Catalogues:** the results of processing imaging data, collecting together a formatted catalogue of astronomical sources and associated parameters.
- **Spectroscopic and associated data:** measurements of target objects in the form of spectral datasets, distance information, chemical composition and physical fields.
- **Studies in the time domain:** observations of moving objects, variable and transient sources requiring multiple observations at different epochs, or synaptic surveys.

- **Theory-based Numerical simulations:** typically spatial data with properties such as position, velocity, mass, temperature, and many others, which may also include a time domain through *snapshot* outputs at different epochs.

A common factor with much of this data is that it may be represented as a multi-dimensional spatial volume. As such, a typical approach for visualisation is *volume rendering*, a set of techniques to project a volumetric data set to a 2 dimensional image. There are several incentives for exploring physical realism in the context of volume rendering, such as generating virtual observations (creating replica imaging data from numerical simulation) and visual exploration of spectral domains where proper treatment of multi-frequency emission is particularly important (further detailed in Chapter 2). Working directly with astronomers, it may be possible to exploit the astronomers knowledge of the data, and source data itself as generated by observation and simulation, to physically motivate and improve volume rendering for astronomical data. As such, the first research question addressed by this thesis is:

How can domain expertise in astronomy be used to physically motivate volume rendering of astronomical data?

Chapter 2 will focus on this question and present improved methods for physically motivated volume rendering of astronomical data, particularly focusing on multi-spectral data sourced from observation and particle based data produced by the N-body and SPH simulations introduced in Section 1.1.

1.3.2 Heterogeneous High Performance Computing

High performance scientific visualisation typically requires some form of high performance computing resources to visualise big data. A computing centre may have a dedicated visualisation cluster exploiting GPUs, or equip a small subset of compute nodes with GPUs in a larger HPC system, to exploit hardware accelerated rendering. Where GPU hardware is not available, as is common for many large HPC systems, OpenGL based visualisation software relies on software-implemented drivers such as Mesa (Mitra and Chiueh, 1998). Such drivers are traditionally much slower than hardware accelerated rendering and not conducive for large scale interactivity (Ahrens, Lo, et al., 2008), although recent work has aimed to improve this.

There is a more recent trend, termed *software defined visualisation* (Jeffers, 2016), focused on CPU-based rendering for visualisation. This trend reflects a paradigm of heterogeneity becoming dominant in HPC, with modern systems exploiting massively parallel computational accelerators and coprocessors for maximum computational efficiency (discussed further in Chapter 3). In some cases, due to the availability of libraries for general purpose graphics programming unit (GPGPU) programming, the accelerators used are in fact GPUs; as such, they are well suited to exploitation by parallel visualisation software. However, there is an increasing trend of both non-GPU based accelerators, and self-hosted many-core platforms which

do not support, or do not have efficient implementations of, traditional graphics libraries such as OpenGL. This means it is necessary to find non-GPU based approaches for visualisation; software defined visualisation has led to developments such as the addition of a high performance software renderer in Mesa (OpenSWR[3]), and other software-based approaches such as the OSPRay software ray-tracer (Wald et al., 2017). However, there is still a significant distance to go for software-defined visualisation approaches to become the norm for domain specific or specialised visualisation tools. Effectively utilising emerging highly parallel hardware requires a significant shift in the way high performance algorithms are implemented, and next-generation visualisation approaches must take into account their presence, noted by Ahern (2012) as part of the high performance visualisation field survey of Bethel, Childs, and Hansen (2012).

In astronomy, high performance computing resources are commonly used to generate and process big datasets. As such, astronomers are aware of the need to exploit heterogeneous systems. As a prime example, the cosomological simulation code of Potter, Stadel, and Teyssier (2017), discussed in Section 1.1.2, is a fully GPU accelerated simulation code. There are many other examples of astronomers targeting heterogeneous performance (Fluke et al., 2011; Malladi, Dodson, and Kitaeff, 2012; Wang and Harris, 2013; Wang, Cao, et al., 2018), due to this heterogeneity trend in HPC system architectures (discussed further in Chapter 3), which prompts the second research question of the thesis:

How can high performance visualisation cope with modern heterogeneous high performance computing systems?

Chapter 3 focuses on this question, in particular examining existing and emerging high performance architectures and identifying key features to focus on to achieve performance on current and future platforms, and addressing the distributed memory hierarchy for volume rendering.

1.3.3 Remote Visualisation in the Modern Research Environment

Sections 1.3.1 and 1.3.2 focus on the quality and performance of high performance visualisation tools, however such tools must also be usable within the *modern research environment*. A research environment encompasses the software tools, platforms, and hardware used for scientific research. This environment is constantly evolving, driven by advancements in hardware platforms, operating systems, software platforms and applications development. Over the last decade, a significant part of this evolution has been driven by maturing of web technologies and the rise of *cloud computing* (Wang, Tao, et al., 2008; Marinescu, 2013). Cloud computing is generally separated into three layers: Infrastructure As A Service (IaaS), Platform As A Service (PaaS), and Software As A Service (SaaS) (Patidar, Rane, and Jain, 2012). IaaS approaches such as Amazon Web Services (AWS) and Microsoft Azure

[3] http://openswr.org/

have brought large scale compute resources to users on-demand, whilst PaaS and SaaS approaches have been used in combination with emerging web technologies to build a great variety of web-based platforms, from private institutional clouds to global research platforms publicly hosting web applications and massive scientific data repositories. Furthermore, the advent of *virtualised computing* solutions such as Docker[4] are impacting methods for application deployment and access to scientific software (Gerlach et al., 2014; Gorgolewski et al., 2017), as well as supporting important topics such as reproducibility of research (Cito and Gall, 2016).

As a result, in many fields the modern domain scientist is able to perform a significant portion of their scientific workflow via the web. However, when it comes to large scale scientific computing, a core enabling technology is traditional high performance computing. Whilst cloud computing is growing in importance, it is not yet a replacement for high performance computing (as found by an early two year feasibility study published by the U.S. Department of Energy (Yelick et al., 2011) and more recent articles (Downing, 2018)). As such, there have been many efforts to incorporate web and cloud-based approaches to traditional HPC and vice-versa (detailed further in Chapter 4). However, the integration of HPC and web technology is challenging, due to the significant disparity between the typical hardware, software, and cultural environments of the web and traditional HPC. A domain scientist developing high performance research software is unlikely to also be an expert in web development, or even familiar with the languages and tools common in the field, and vice versa.

In terms of high performance scientific visualisation, this challenge is particularly relevant, due to the requirement for remote visualisation. High performance computing platforms are inherently remote; the hardware is typically installed in a supercomputing centre, or at a university campus, and users access such a machine remotely via protocols such as Secure Shell or remote desktop software. Running jobs on a remote system traditionally consists of submitting a batch script to a scheduling system containing a series of commands to run an application. Conversely, scientific visualisation is more naturally an interactive process. Exploratory visualisation may involve manipulating the view (for example, rotating or zooming), filtering or modifying the dataset, changing visualisation parameters and many other interactions. As such, remote visualisation approaches are necessary to converge the interactivity of visualisation with the remote nature of HPC.

Typical approaches to remote visualisation employ a client-server paradigm, relying on remote desktop software and/or bespoke desktop clients installed on the user machine that are connected to visualisation servers running remotely on a HPC system. However, in view of the move towards web platforms in the modern research environment, it is logical that a new remote visualisation approach should be

[4] https://www.docker.com/

web-capable. Whilst there are a variety of existing approaches to remote visualisation, there are very few that can effectively integrate with the modern web environment.

For astronomers facing big data challenges, visualisation is by necessity a remote task. A web-based approach can support the development of accessible tools to integrate with scientific work flows and environments, a key difficulty for astronomical visualisation as highlighted by Hassan and Fluke (2011), and the focus of Section 1.3.4. As such, the third research question for this thesis is:

How can modern astronomical tools exploit remote and interactive high performance visualisation on the web?

Chapter 4 focuses on this question, aiming to bridge the boundaries of web and high performance computing, supporting the integration of such disparate environments in order to create a visualisation tool that can be fully exploited within the modern research environment.

1.3.4 Access and Analysis via the Web

As discussed in Section 1.1, the rising complexity and size of scientific datasets has caused astronomers to face increasingly difficult processes of data storage and access. These difficulties can be exacerbated by logistical problems relating to long term maintenance, and can create barriers that prevent data sharing and inhibit collaboration. Physical storage and transport, security protocols, and complex file formats are just some of the problems that can prevent researchers from effectively managing their data and lead to an environment that is not conducive to cross-institutional and international collaboration. For example, in the case of cosmological N-body simulations, the resultant datasets are typically stored on parallel file systems at the HPC centre hosting the resources used to run the simulation. The size of such datasets, and the computing capacity required to post-process them, means this is often the most economical place for storage. However, other researchers may not have access credentials or computing resources to process the data at the host computing centre. This can be a important issue, as significant simulation results are relied upon by many researchers for a wide range of scientific purposes. As an example, the well-known flagship Millennium cosmological simulation paper (Springel et al., 2005) has been cited over four thousand times so far, many of which are works utilising the Millennium database.

It has been commonplace for data-intensive scientists to employ online databases and associated web portals for many years now in an effort to address this data accessibility issue (Szalay and Gray, 2006). Astronomers have been leading the way with concepts such as the Virtual Observatory (Djorgovski and Williams, 2005), and e-research platforms such as the Strasbourg Astronomical Data Center (CDS)[5]. For large theory datasets, a solution that is gaining in popularity is web-based. There

[5] http://cdsweb.u-strasbg.fr/

is now emerging a new type of web-portal for theory simulations, that not only provides access to the raw outputs of simulation, but includes tools to assist the astronomer in data exploration and creating or accessing derived datasets more directly useful for their science case. Recent examples are also supported by powerful hardware back-ends (e.g. HPC), enabling the processing of large and complex datasets through sophisticated on-line services.

Such high performance back-ends can be a necessity in order to support tools for selection, customization, and generation of derived data to suit the requirements of a specific science case. In this context, visualisation tools can be an effective support for analysis, helping the scientist to find the subset of data they require. For example, it is already commonplace to exploit visualisation in virtual observatories; while searching for and extracting observational data objects, a user typically visualises and navigates images in their web browser using an interactive 2D interface such as the Aladin Lite Sky Atlas.

Ideally, a user should be able to follow a similar interactive navigation process to access theoretical data. However, data from computer simulations is frequently expressed in three spatial dimensions, and more naturally explored via interactive 3D visualisation. This is a more difficult service for a web portal to provide because on-demand 3D visualisation can be computationally expensive, especially so in this context as the size and complexity of theoretical data can easily exceed the capabilities of modern web browsers, and is continually growing. This problem motivates the final research question:

How can remote high performance visualisation facilitate access and analysis of large astronomical datasets on the web?

Chapter 5 focuses on this question through collaboration with domain experts to incorporate high performance visualisation in typical web-based astronomical analysis scenarios.

1.4 Splotch as a Software Context

Scientific visualisation research needs a software context within which to be implemented. This work utilises Splotch, an existing high performance scientific visualisation code, as such a context. Splotch (Dolag, Reinecke, et al., 2008) is a scientific visualisation package designed to run on HPC systems and process large particle-based datasets. Splotch is written entirely in C++, with minimal dependencies beyond those for parallel models and specific file I/O. Splotch can exploit OpenMP for shared memory parallelism, MPI for distributed memory parallelism, and has already begun to address heterogeneous machines with computational accelerators (Jin et al., 2010) (Rivi et al., 2014). Further more, Splotch is fully open-source, providing an accessible code base in which to make developments.

These qualities, as well as the fact that it is one of the few parallel visualisation codes focusing primarily on particle data for astronomy, make Splotch an ideal software context for this research based on the research questions presented:

1. Physically Motivated Particle Visualisation

   The Splotch visualisation pipeline is customised particularly for astronomical particle data, and the rendering component is based on a variant of the volume splatting algorithm using points as the primary geometric primitive, which provides a good starting point to address this motivation.

2. Heterogeneous High Performance Computing Architectures

   The nature of Splotch, written in C++ with few dependencies, means it is amenable to algorithm porting without needing to address the issues of porting dependent libraries or finding/writing alternatives. Furthermore, the software already targets high performance architectures with some addressing of heterogeneity, providing a good starting point to address this motivation.

3. Remote Visualisation in the Modern Research Environment

   Splotch was initially designed as a simple and lightweight code, in order to build on a variety of architectures and efficiently visualise large datasets. As such, the small code-base is amenable to the modification and extension necessary to build a remote visualisation tool.

4. Access and Analysis via the Web

   The simple and lightweight nature of Splotch also make it suitable for embedding in other software and so it provides a good candidate to integrate with existing scientific workflows. An example of this is the embedding of Splotch already within VisIVO, a larger scientific visualisation package. Furthermore, the astronomical focus of Splotch is ideal for embedding within astronomy focused web platforms.

An introduction to the Splotch code will be presented in Chapter 2, and will be further detailed throughout this work.

1.5 Research Questions and Objectives

To summarise the concepts presented in this chapter: the increasingly challenging paradigm of Data-Intensive Science requires new and evolved tools for analysis. In the context of scientific visualisation, this includes improved visual analysis approaches, such as physically motivated rendering of multi-spectral and particle based astronomical datasets (Section 1.3.1). Looking towards future hardware platforms, there is a growing trend of heterogeneity in modern high performance computing systems; to realise the performance potential of these systems for next

generation tools we must understand how to address these platforms for high performance visualisation (Section 1.3.2). Furthermore, to utilise such systems and process big scientific data, remote visualisation approaches are absolutely necessary; but in order to be useful to the scientist, such approaches must enable interactive use within the context of the modern web-enabled research environment. This requires understanding the general remote approaches for HPC applications and how to integrate such approaches with web technologies (Section 1.3.3). Finally, the size and complexity of scientific data is challenging for scientists aiming to collaboratively analyse and share their data. The integration of visualisation tools to web research platforms for data analysis and sharing may be able to help alleviate this difficulty and improve the accessibility of data for the modern research scientist (Section 1.3.4).

This thesis will aim to answer the four research questions posed in Sections 1.3.1 to 1.3.4 through four key objectives to be accomplished within the thesis. These objectives are to:

O.1 Understand and exploit domain specific data and knowledge to physically motivate volume rendering for astronomical data. (***Q.1***)

O.2 Analyse and exploit emerging HPC architectures in the context of high performance visualisation. (***Q.2***)

O.3 Explore a general approach to addressing web-based interactive high performance visualisation. (***Q.3***)

O.4 Discover if and how high performance visualisation can support astronomers in typical web-based access and analysis scenarios. (***Q.4***)

Objective **O.1** is addressed in Chapter 2, starting from an extended volume rendering approach in Splotch, which is exploited in a novel galaxy modelling and visualisation pipeline, and concluding with a study of physical motivation via radiative transport for astrophysical simulations.

Objective **O.2** is addressed in Chapter 3 by understanding the current state and future directions of HPC systems, and identifying implementation and optimisation paths for emerging and distributed memory high performance architectures.

Objective **O.3** is addressed in Chapter 4 through a web based remote interaction framework for high performance applications, which is then utilized to turn batch volume rendering algorithm Splotch into an interactive, web-based, high performance volume rendering tool.

Objective **O.4** is addressed in Chapter 5 through a prototype application of the new interactive, web-based, visualisation tool as a means of accessing and manipulating data within Theoretical Virtual Observatories for astronomy.

The ordering of research questions and objectives is defined such that each chapter relies on the work or context of the previous chapters. Figure 1.4 illustrates how these objectives come together to form a concerted visualisation approach. The

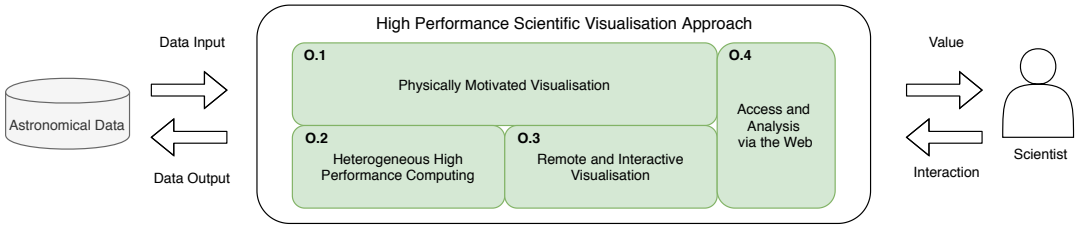


FIGURE 1.4: An overview of the research presented in this thesis, showing how the related objectives fit together to form a high performance visualisation approach to help the scientist extract value from their scientific data.

visualisation approach of Objective **O.1** is underpinned with effective exploitation of emerging HPC architectures (**O.2**) and modern remote visualisation approaches (**O.3**). These are combined with approaches to support data access and analysis (**O.4**) to present the scientist with an effective and interactive tool for extracting value from their scientific data.

The remainder of the thesis is organised as follows: Section 1.6 provides a brief summary of the work contained in the thesis, Chapters 2 to 5 make up the core research chapters of the thesis, addressing each research question and objective as detailed above. Finally Chapter 6 summarises the overall research contributions and presents a series of avenues for future work.

1.6 Thesis Summary

The research work presented in this thesis addresses the overarching problem of big data in astronomy through these four objectives, dedicating a chapter to each respectively to answer the four posed research questions. A brief research summary for chapter is presented below.

The question, *'How can domain expertise in astronomy be used to physically motivate volume rendering of astronomical data?'*, is concerned with realism and the physical motivations for particle visualisation in astronomy. Chapter 2 begins by introducing the concepts of realism in visualisation and providing an overview of the physical basis of volume rendering, including optical models, transfer functions and typical algorithmic approaches. The optical model of Splotch is explained, and an extension presented to better support independent emission and absorption coefficients and multi-wavelength data for astronomy. A novel pipeline for 3D reconstruction and visualisation of galaxies based on observational image data, exploiting this extension to add an absorptive dust component. Finally the work of this chapter moves beyond the visualisation to present a radiative transport algorithm implemented for astrophysical simulations, highlighting the potential for further physical motivations to Splotch based on approaches used in astronomy.

The question, *'How can high performance visualisation cope with modern heterogeneous high performance computing systems?'*, is concered with the growing complexity

of HPC systems the difficulties associated with effectively exploiting them. Chapter 3 first provides an overview of the history of high performance computing systems, focusing on the hardware trends from a software optimisation point of view to identify a series of features that should be targeted for high performance software optimisation. These are then explored in the context of multiple types of many-core, accelerator, and distributed memory hardware, providing a series of experiences and examples of effective optimisation on heterogeneous high performance architectures. The experiences are then collated into a series of methodologies for algorithm optimisation on future architectures.

The question, *'How can modern astronomical tools exploit remote and interactive high performance visualisation on the web?'*, addresses the growing diversity of the modern research environment, which exploits web-enabled applications and diverges from the traditional environment of high performance computing. Chapter 4 presents a categorisation of remote applications for HPC, and identifies a lack of software for integrating high performance applications with web environments. To combat this, a framework for remote and web-based interaction with high performance applications is presented. The framework, WSRTI, is then utilised to build an interactive remote interface to Splotch, in an effort to support web integration of high performance visualisation tools for research environments.

The question, *'How can remote high performance visualisation facilitate access and analysis of large astronomical datasets on the web?'*, focuses on the need to make large theoretical datasets accessible for the wider astronomical community. The introduction of web portals supported by high performance resources provides an excellent opportunity for 3D visualisation to support data access and sharing. In particular, Chapter 5 surveys the visualisation capabilities of existing portals, and introduces a prototype web visualisation approach based on the work of previous chapters for integrating with such portals. The Theoretical Astrophysical Observatory is exploited as an example use case, with a prototype implementation focusing on quantitative visualisation and filtering mechanisms. The utility of such a tool to support the data access model of theoretical virtual observatories is finally illustrated through demonstrative examples of astronomical workflows that could be augmented.

The research presented in this thesis looks towards the future of data-intensive scientific computing, and seeks to support research scientists through high performance 3D visualisation techniques. Each of the chapters presented in this thesis are supported to varying degrees by academic publications, as described in each respective chapter introduction. Throughout this work, the high performance visualisation package Splotch is built into an interactive, remote, web-based visualisation solution for astronomers, with the intention that Splotch can be a tool to support scientific analysis in the emerging environments of large scale astronomy. In addition to the specific research contributions described above, Splotch is fully open-source and the work presented here will be integrated to the public Splotch code repository for presentation to the astronomical community.

Chapter 2

Physically Motivated Particle Visualisation

This chapter aims to answer the question **Q.1**: *'How can domain expertise in astronomy be used to physically motivate volume rendering of astronomical data?'*, by addressing the corresponding objective **O.1**: *'Understand and exploit domain specific data and knowledge to physically motivate volume rendering for astronomical data'*. The chapter begins with a discussion of visual representations and realism in computer graphics, highlighting the relevance for astronomical visualisation (Section 2.1). The radiative tranport underpinnings of volume rendering are introduced, and used to analyse the existing rendering algorithm in high performance particle visualisation code, Splotch (Section 2.3). An improved rendering model is presented to better support physical realism for astronomical particle visualisation and used for the addition of dust lanes within a novel galaxy modelling and visualisation pipeline (Section 2.4). The synergy between physically realistic volume rendering and physical simulations in astronomy is demonstrated through a ray-casting based radiative transport algorithm for astrophysical simulations, identifying areas of knowledge transfer for future work (Section 2.5.1). The chapter concludes with a brief summary of research contributions (Section 2.6).

2.1 Physically Motivated Visualisation

As introduced in Section 1.3.1, realism in visual representations can be distinguished as physical realism, photorealism, and functional realism. There are a variety of application areas in which physical realism is important for volume rendering (Rushmeier, 1995), and in computer graphics literature there has been extensive study of advanced participating media rendering techniques in military, industrial, commercial contexts as documented by the comprehensive survey of Cerezo et al. (2005). In the context of astronomy, there are existing examples of physically motivated approaches to scientific visualisation. For example, Magnor, Kindlmann, et al. (2004) introduced a modelling and physically realistic visualisation approach for reflection nebulae that is further built on in (Magnor, Hildebrand, et al., 2005). Kaehler et al. (2006) describe a GPU-assisted volume rendering tool for adaptive mesh refinement

cosmological simulations modelling frequency specific emission, which was further extended to global illumination in (Ralf Kaehler, 2013). SPLASH (Price, 2007) is a particle-based astronomical visualisation tool targeting SPH simulation outputs which takes special care to physically reconstruct SPH kernels for rendering. Whilst primarily for entertainment, the black hole visualisation for the recent movie Interstellar was also highly physically motivated (Thorne, 2014), leading to new insights on light transport near to black holes. Further examples are highlighted in the survey of Magnor, Sen, et al. (2010) on modelling and rendering techniques for digital planetariums.

A particular area of interest in astronomy that requires special consideration for visualisation is representing the wider spectral domain. Astronomical phenomena emit radiation across a broad range of the electromagnetic (EM) spectrum; observed images, of a single galaxy for example, are typically captured many times at different frequencies to build a comprehensive understanding of the galaxy's intrinsic physical characteristics (Section 2.4 will expand on this with examples). This multi-frequency emission is interesting for visualisation in multiple ways, for example:

- Spectral cube analysis

  Spectral cubes are three dimensional image arrays, where the third dimension is frequency. Such cubes are increasingly produced by modern astronomical surveys, and can represent complex velocity or kinematic structures (e.g. as used in the kinematic modelling of Section 2.4.3). Hassan, Fluke, et al. (2013) explore the volume rendering of large astronomical spectral cubes, and discuss the applications of generic transfer functions for informing source finding techniques in radio astronomy spectral cubes. More recently, Vohl, Fluke, et al. (2017) develop a series of novel transfer functions to help intuitive visual exploration of spectral cubes through volume rendering.

- Virtual observations of theory data

  Virtual observation is an established means of generating simulated images of theory data for comparative visualisation and for testing pipelines for future surveys, providing an important connection between computational and observational astronomy. Virtual imaging tools, such as Phox (Biffi et al., 2012) or SkyMaker (Bertin, 2009) are essentially physically realistic visualisation tools, tailored to generate frequency-specific images and often replicating a specific telescopic instrumentation to enable quantitative analysis and comparison with observed images.

- Visual exploration of multi-frequency data

  In general, the dimensionality of multi-frequency data can pose challenges for visualisation, and tends to require novel transfer functions or presentation

methods to display and differentiate the variety of data characteristics. For example, the work of Li, Fu, and Hanson (2008) explores a variety of volume rendered approaches to present and explore multi-frequency astronomical imaging data. Frequency-dependency is also important for physically motivated visualisations of simulation data, for example using the common UBVRI filters Bessell (2005) shown in the work of Kaehler et al. (2006).

Whilst the examples given here show interesting and novel uses of advanced rendering techniques and physical motivations, they are very much the exception rather than the norm; there is not a large quantity of research in multi-frequency and physically motivated scientific visualisation for astronomy. Imaging tools (such as Phox and Skymaker) are focused completely on physical realism, lacking features for exploration such as interactivity, whilst there is only minimal research focusing on physically motivated, interactive, and explorative visualisation using existing volume rendering tools. As the size of simulation data increases, along with associated particle-based data such as galaxy catalogues, the need for fast and intuitive visual analysis will only become larger. An example of the potential for visualisation tools, discussed further in Chapter 5, is using frequency-dependent volume renderers as pseudo-imaging tools for interactive analysis, allowing users to interactively select and filter data before applying more robust, and computationally costly, imaging methods.

2.2 Volume Rendering from a Physical Perspective

For data represented as a volume in 3 dimensional space, as typically found in astronomy, volume rendering can be split into *direct* and *indirect* approaches.

- **Indirect Volume Rendering (IVR)**:

  Indirect approaches typically necessitate the extraction of an intermediate representative geometry from volumetric data. This geometry is then rendered using appropriate graphics rendering techniques or libraries. A common example of this is the extraction and rendering of an isosurface (e.g. the *Marching Cubes* algorithm (Lorensen and Cline, 1987)).

- **Direct Volume Rendering (DVR)**:

  Direct approaches render volumetric data explicitly, without converting to an intermediate representation. This class of algorithms typically aim to build an image by approximating *radiative transport* to replicate the effect exacted upon light rays as they pass through a volume of material (or a *participating medium*).

Whilst IVR techniques can be effective, they require extracting a set of surfaces to reasonably represent the volume, which may not be practical or even feasible, particularly for cloud-like and smooth-varying fields as discussed by Meissner et al.

(2000). The remainder of this work will focus on direct volume rendering approaches based on radiative transport, which are well suited to the large, complex, and often gaseous nature of astronomical data.

2.2.1 Radiative Transport

In the context of volume rendering, physical realism is typically addressed by accurate simulations of light transport in participating media, and physically realistic choices for the parameters of such simulations. This is achieved by solving the well-known radiative transfer equation, which describes the variation of intensity for a ray of electromagnetic radiation passing through a participating medium (as detailed in Rybicki and Lightman (1979), for example). As such a ray passes through the medium, energy is either added through *emission*, removed through *absorption*, and further by photon *scattering* in and out of the ray. Modelling such transport allows a rendering algorithm to approximate the intensity and location of light reaching a virtual camera placed in the scene, and the different approximations of radiative transfer form the well known optical models of volume rendering introduced by Max (1995).

Presented here is the basis of one of the most common approaches for volume rendering (Max and Chen, 2010; Jönsson et al., n.d.); an optical model considering emission and absorption. Discussion on more advanced optical models, such as those including illumination effects can be found in Section 2.3.4.

The formal radiative transfer equation in integral form can be presented as following (e.g. Rybicki and Lightman (1979)):

$$I_v(x_1) = I_v(x_0)\exp(-\tau_v(x0, x1)) + \int_{x0}^{x1} j_v(x)\exp(-\tau_v(x, x1))dx \tag{2.1}$$

This equation describes the *intensity* I of radiation in frequency v at location $x1$ along the ray, as a function of the radiation at location $x0$ and the frequency dependent emission or absorption effects of the intervening medium, illustrated in Figure 2.1.

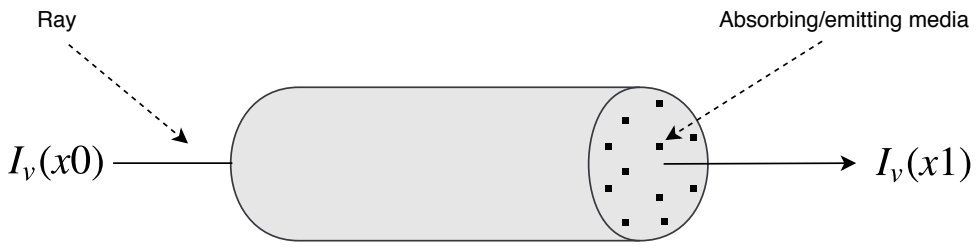


FIGURE 2.1: An illustration of the physical process described by Equation 2.1. A ray with initial intensity $I_v(x0)$ in frequency v passes through a participating medium which absorbs from, or emits energy to, the ray for a resulting intensity $I_v(x1)$.

In equation 2.1, j represents the *emissivity* of the volume, $\tau(x0, x1)$ represents the *optical depth* between locations x_0 and x_1. The optical depth describes the likelihood of the average photon being absorbed while passing through the medium, and is defined as:

$$\tau_v(x0, x1) = \int_{x0}^{x1} \alpha_v(x)\, dx \tag{2.2}$$

where α is commonly described as:

$$\alpha_v = \kappa_v \rho(x) \tag{2.3}$$

with α_v representing the *absorption coefficient* in frequency v. κ_v is known as the *opacity* or *mass absorption* coefficient, also dependent on v, and ρ is the mass density at position x. An optical depth $\tau_v > 1$ is considered *optically thick* or opaque, whilst $\tau_v < 1$ is considered *optically thin* or transparent.

The two terms on the right hand side of equation 2.1 can be conveniently separated into two approximations of radiative transfer that form separate optical models as described by Max (1995). The first term represents *absorption* of the initial intensity, and in solitary defines an absorption only optical model, i.e. an absorptive medium that does not scatter or emit light:

$$I_v(x_1) = I_v(x_0) \exp(-\tau_v(x0, x1)) \tag{2.4}$$

I.e. the intensity at location x_0 diminished by absorption of the medium. The intensity at point $x1$ along a viewing ray is equal to the initial intensity at $x0$, attenuated by the optical depth of the intervening medium. As described by Max (1995), such a model can be used effectively for X-ray images, or simple models of absorptive media such as black smoke.

The second term represents true emission, which in solitary defines an emission only optical model, i.e. an emissive medium that does not absorb or scatter light:

$$I_v(x_1) = I_v(x_0) + \int_{x0}^{x1} j_v(x) dx \tag{2.5}$$

Here the intensity at point x_1 is equal to the initial intensity at x_0 plus the cumulative contribution of medium emission j.

The following sections provide an overview of the more practical aspects such as the definition of optical properties (j_v, κ_v) and algorithmic approaches commonly employed.

2.2.2 Transfer Functions

The transfer function defines how data characteristics are mapped to opacities and colours during visualisation. From a physical perspective, these define the emission

and absorption coefficients (j, σ) used during radiative transport. The definition of a suitable transfer function is a difficult problem in volume rendering, and the focus of much research (e.g. Pfister et al. (2001) and Arens and Domik (2010)). A recent State-of-the-Art Report by Ljung et al. (2016) presents a thorough overview of the existing literature and current research challenges for transfer functions. A typical approach is to evaluate considering functional realism, i.e. mapping data quantities of interest to Red, Green, and Blue (RGB) image pixel components to best convey the data features of interest.

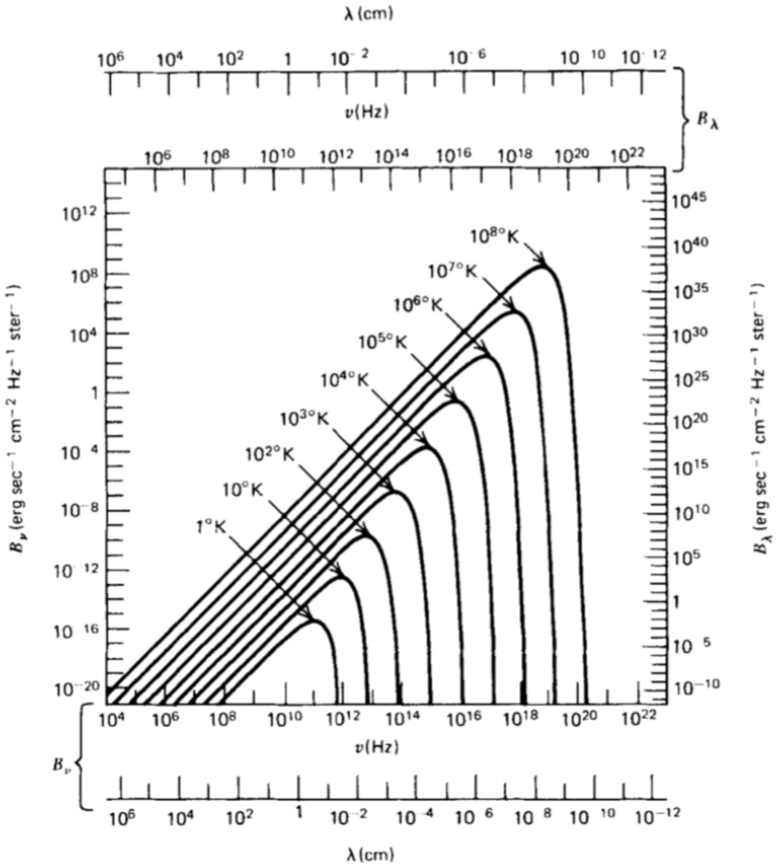


FIGURE 2.2: An illustration of the Planck Law described by Equation 2.6. The spectrum of emission (B_v) across frequency and wavelength for variety of temperatures, as illustrated by (Rybicki and Lightman, 1979).

A physically motivated transfer function should consider the material characteristics that influence emissive and absorptive coefficients, including awareness of frequency dependencies. For example, Planck's law relates temperature to the spectral density of thermal radiation emitted by a black body (or a *perfect* emitter, whose emission is not dependent on location or composition). This is a typical approximation describing the thermal stellar and gas emission in astronomical simulation, the law is defined as following:

$$B_v(v, T) = \frac{2hv^3}{c^2} \frac{1}{e^{\frac{hv}{k_B T}} - 1}. \tag{2.6}$$

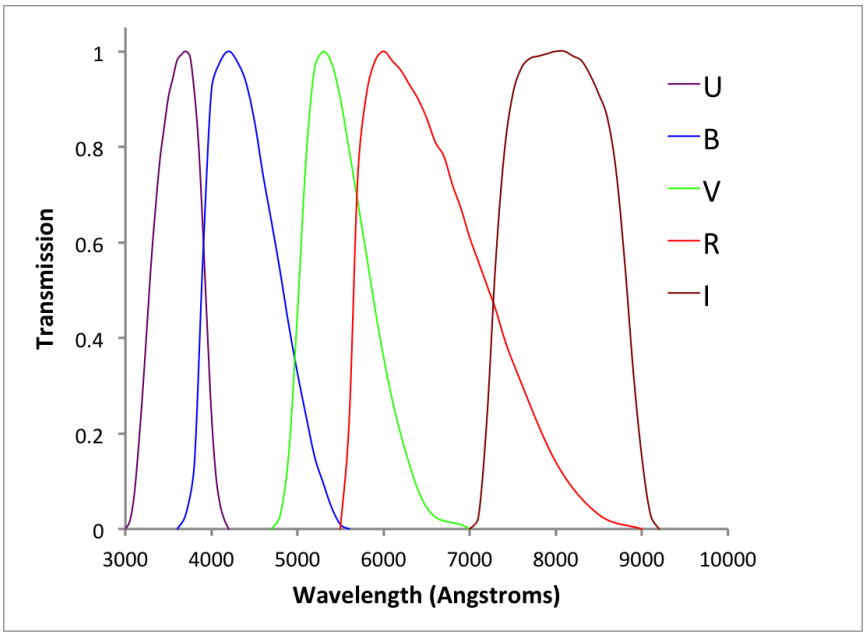


FIGURE 2.3: An illustration of the Johnson-Cousins UBVRI optical filters, commonly used in astronomical observations (Bessell, 2005).

where B_v is the emissive intensity as a function of frequency v and temperature T, h is the *Planck constant*, c is the speed of light, and k_B is the *Boltzmann constant*.

B_v approximately describes the amount of energy emitted by an object (blackbody) via thermal radiation. Figure 2.2, from (Rybicki and Lightman, 1979), illustrates the spectrum of Planck emission for such a blackbody at a variety of temperatures. For astronomical visualisation, one may use a transfer function based on B_v to approximate observations using frequency dependent coefficients and allow, for example, comparative visualisation with observed images.

To illustrate how this may be done, one may apply a series of optical filters such as the Johnson-Cousins UBVRI filters (as shown in (Bessell, 2005)) shown in Figure 2.3. This well-known photometric system is one of many standardised systems of filters (also known as *passbands*) used for capturing observational images in astronomy, the standardisation of which allows quantitative analysis and comparison between instruments (a full review of such systems is provided by (Bessell, 2005)).

A convolution of the true emission, as illustrated in Figure 2.2, and a series of filters such as shown in Figure 2.3, shows a suitable example of physically motivated emission coefficients for specific frequency ranges. An example of this applied in astronomical visualisation can be found in the work of Kaehler et al. (2006). It should be noted this example serves only for thermal radiative emission; there are other types of emission and absorption that can be considered which are governed by separate physical processes, for example emission of neutral atomic hydrogen in radio frequency ranges (the *21cm hydrogen line*) which can penetrate the interstellar dust clouds that typically absorb visible light.

2.2.3 Data Representations

There are a variety of algorithms approximating radiative transfer used in volume rendering, which are often described in terms of the data structures on which they are designed to operate; as such, to contextualise such algorithms, the typical data structures used in Astronomy must first be introduced. In a volume rendering context, these data structures can be effectively categorised in terms of the geometric structure of the data, or how data is discretised in the spatial domain. The most common structures are summarised below, based on the work of Lang and Grave (1993):

- **Structured Grid**: Also known as a regular grid, this is a grid of locations defined by a regular connectivity, i.e. each point has the same number of neighbours. There are a variety of structured grids, for example:
  - *Cartesian*: Coordinates are not explicitly stored, each index is a unit measurement along a Cartesian axis.
  - *Uniform*: Typically rectangular, this grid is defined by a constant spacing between grid lines (although distinct between axes). Also known as a regular grid, coordinates are not typically stored and can be obtained via a bounding box, definition of the grid spacing, and indexing scheme.
  - *Rectilinear*: A more general case of the uniform grid, this is also usually rectangular but allows a non-uniform spacing along each coordinate axis. This is described by a grid vector for each coordinate axis.
  - *Curvilinear*: A further generalisation, where each cell is a more general quadrilateral or cuboid shape.

  Such methods are typically used in image processing (pixel arrays), and CFD codes exploiting Eulerian fluid flow techniques Fletcher (1988).

- **Unstructured Grid**: Also known as an irregular grid, each point has both an explicit location and connectivity information stored, without a uniform definition across the grid. Typically used for finite element methods, e.g. Morgan and Peraire (1998).

- **Adaptive Grid**: Also known as a block-structured or multi-domain grids, this is characterised by a patchwork of grids at differing resolutions. Used in adaptive mesh refinement techniques for applications such as adaptive physics simulations with regular grids (e.g. Teyssier (2002)), and may also be applied to unstructured grids (e.g. citetJahangirian08).

- **Scattered Points**: Also referred to as particle data, this is defined as a collection of points with location explicitly stored (e.g. 2D or 3D spatial coordinates) without a geometric relationship between the points. Also referred to as particle data, each point usually has fields associated with it such as temperature,

density, etc. As an example, this structure is typically used in CFD codes exploiting Lagrangian fluid flow techniques such as Smoothed Particle Hydrodynamics (Gingold and Monaghan, 1977).

The focus of this work is primarily on scattered point or particle data for astronomy, a popular data structure used by N-body and SPH simulations as described in Section 1.1.2. For a review of simulation techniques and the data structures required see, for example, Dolag, Borgani, et al. (2008). In general, the structures presented in this section can be mapped to the sources of data in astronomy as introduced in Section 1.3.1, as shown in Hassan and Fluke (2011).

2.2.4 Algorithmic Approaches

There are a variety of algorithmic approaches to computing radiative transfer in computer graphics, with varying levels of approximation. For a comprehensive overview of such approaches, the reader is referred to the early surveys of (Elvins, 1992) and (Meissner et al., 2000) focusing on general volume rendering algorithms; (Cerezo et al., 2005) focusing on volume rendering illumination methods; (Marmitt, Friedrich, and Slusallek, 2008) on CPU-based techniques with an emphasis on ray-tracing; and (Beyer, Hadwiger, and Pfister, 2015) focusing on large scale GPU-based approaches. An important distinction for such algorithms is whether they are *object-order* or *image-order* (Lacroute and Levoy, 1994), also known as *forward mapped* and *backward mapped* (Zwicker et al., 2001), or *feed-forward* and *feed-backward* (Westover, 1991) respectively. This distinction defines whether the data is mapped to the image plane (object order), or the image is mapped to the data plane (image order). Practically this can be illustrated, as in (Marmitt, Friedrich, and Slusallek, 2008), by the pseudocode in Figure 2.4.

```
for each pixel in image:
  cast ray; identify intersecting primitives
  for each primitive intersected:
    accumulate contributions
```

```
for each primitive in scene:
  identify intersecting rays
  for each ray intersected:
    accumulate contributions
```

FIGURE 2.4: The basic pseudocode representation of image-order rendering (left), v.s. object-order rendering (right).

A brief overview of some of the most popular algorithms is given here:

- **Cell Projection**

  Typically an object-order approach, the cell projection group of algorithms focuses on rendering unstructured grid data. The cells of, for example, tetrahedral meshes are decomposed into triangles and projected to the image, with colours computed at triangle corners and interpolated across faces. (Marmitt, Friedrich, and Slusallek, 2008) provide a detailed overview of the variety of research efforts focused on cell projection.

- **Volume Splatting**

  Also an object-order approach, volume splatting is sometimes known as *vertex projection*. First introduced by (Westover, 1991) focusing on structured grids, this approach has applicability to most data representations containing vertices. Each data sample is projected to the image and spread using a footprint function, commonly a Gaussian kernel, forming a 'splat' on the image. Such splats are composed back to front with respect to the image plane, or via a sheet buffer approach grouping samples parallel to the image plane (Mueller and Crawfis, 1998).

- **Texture Mapping**

  An early survey of the capabilities of texture mapping is presented by Heckbert (1986), and the introduction of hardware support on GPUs further encouraged this approach; one of the earliest uses for volume rendering was presented by Cabral, Cam, and Foran (1994). This image-order approach generates image slices (or slabs), by trilinearly interpolating the dataset stored in a 3D texture using hardware supported interpolation, which are then blended to the final image. (Beyer, Hadwiger, and Pfister, 2015) discusses the varying approaches to this technique in detail. The method applies to structured grids, however is a popular GPU based approach due to hardware texture support, so other representations are typically voxelised to take advantage of this method (e.g. Levenetal02).

- **Shear Warp**

  Introduced by (Lacroute and Levoy, 1994), the shear warp algorithm combines the benefits of object and image order methods. Focused on structured grids, the volume must be resampled to a rectilinear grid. The grid is sliced and sheared in 3D such that the viewing rays are perpendicular to the volume, with an optional scaling for perspective. The slices are composited to a single slice, and finally warped to image space and resampled to produce the final image.

- **Volumetric Ray Casting**

  Volume ray casting is an image order approach that casts a ray into the volume for each image pixel. At typically equidistant points along the ray the volume is sampled, by interpolating from the closest data points. At each sampling point, the transfer function is computed and shading applied. Finally the samples are composited solving the rendering equation to compute the final pixel values. This method applied best to structured grids, simplifying the ray-voxel intersections.

- **Ray Tracing**

Ray tracing is a technique used for surface rendering, however can also be applied to volumetric data where a surface can be extracted. This image order approach casts a ray for each pixel, and computes the first intersection with a surface in the scene. At such an intersection, material properties and lighting properties are considered to compute the pixel colour, which may include illumination by tracing secondary rays to light sources. For volume rendering, iso-surfacing is a typical approach that extracts a surface representing points of constant value in volumetric data (Marco K. Bosma, 1998), allowing ray-tracing to be applied. The survey of Marmitt, Friedrich, and Slusallek (2008) discusses the variety of approaches in detail.

This list describes commonly used algorithms for volume rendering in scientific visualisation, although not exhaustively; there are many hybrid algorithms and variations with differing optical models as detailed in the surveys provided.

The focus of this work is on volume splatting approaches, which are particularly well suited for large astronomical particle data. This is because the particle data structure can also be treated natively as a volume splatting element, which negates the need for re-sampling to a regular grid for other volume rendering algorithms. Such re-sampling can lose the inherent adaptive resolution of particle-based data, an important feature for particle-based simulation methods. The object-order nature of splatting is typically efficient, discarding those primitives outside of the viewing frustum during transformation stages. A variation of the traditional splatting approach tuned for parallel particle rendering is employed by the Splotch software, introduced in the following section.

2.3 A Physical Approach to Particle Visualisation

2.3.1 Splotch

Splotch, as introduced in Section 1.4, is a high performance scientific visualisation software package designed for point-like particle data. The primary focus of Splotch is cosmological N-body and SPH point-like simulation data, which describe fluid flow using tracer particles that are spread across a 3D domain using a kernel such as the B_2-Spline (Monaghan and Lattanzio, 1985), defined such that particles overlap with many neighbours (e.g. typically 32 or 64). A set of typical Splotch outputs are shown in Figure 2.5. This composite of images, courtesy of Klaus Dolag of the Max-Planck Institute for Astrophysics, Garching, showing a zoom in to *Box0/mr* from the Magneticum cosmological simulation suite[1].

Figure 2.6 illustrates the structure of the Splotch code, showing each stage in the visualisation pipeline from binary data to output images.

[1] http://www.magneticum.org

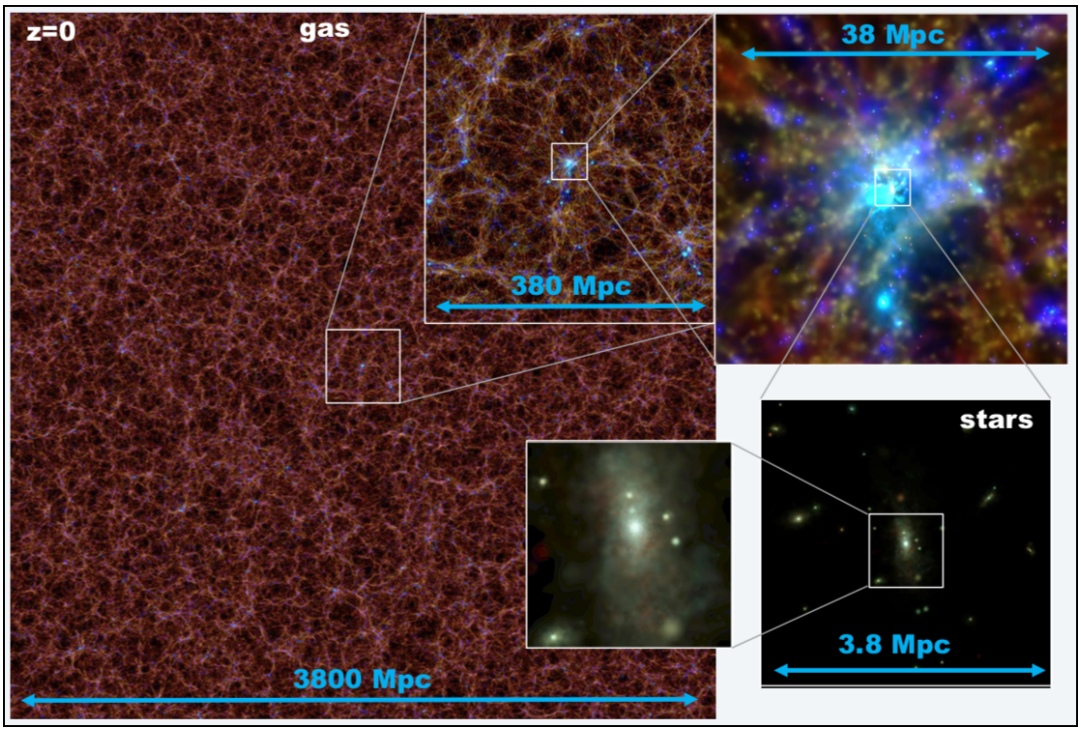


FIGURE 2.5: *Box0/mr* from the Magneticum cosmological simulation suite, courtesy of Klaus Dolag of the Max-Planck Institute for Astrophysics, Garching. The data consists of ∼186 billion particles at $z = 0.0$, representing gas, stars, dark matter and black holes, and the leftmost image was rendered at 16000 × 16000 resolution.

Input: A key-value parameter file is provided by the user, which defines the input file and scene setup (e.g. camera position for single images or paths for movie generation). Data is read using one of a variety of parallel readers specialised for common astrophysical formats, including block and tabular binary files, Gadget, RAMSES, Enzo, Tipsy, HDF5 amongst others; all readers support MPI for parallel input. At this stage the requested fields are extracted and stored in a particle structure based on user parameters.

Project and Colour: This stage performs ranging, normalization, and optionally applies generic functions to input fields (e.g. logarithm). Particle positions represented as 3 dimensional cartesian coordinates are transformed (roto-translated), and a perspective projection is applied with reference to supplied camera and look-at positions, followed by clipping. Colours, stored as a triplet of single precision floating point values representing red, green, and blue components (RGB) are generated either directly from a vector input field (e.g. three component velocity), or as a function of a scalar input field (e.g. single component temperature) using a colour map defined by runtime input parameters. This stage is mostly *embarrassingly parallel*, and parallelised with OpenMP.

Render: The rendering stage consists of a hydrid parallel volume splatting algorithm. An OpenMP threaded approach is applied for shared memory parallelism, in which the output image is discretised into 2D tiles, particles affecting each tile are

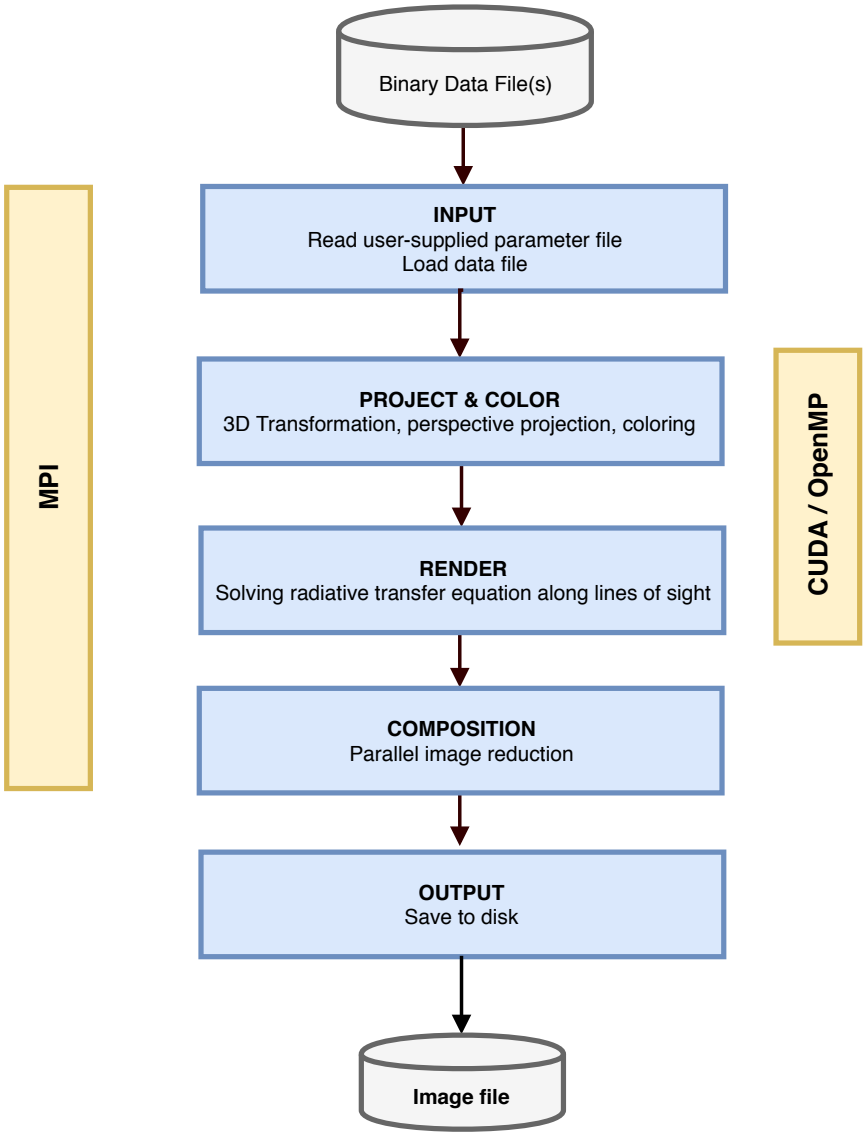


FIGURE 2.6: The Splotch Structure: an overview of the key stages in the Splotch code.

calculated and contributions are accumulated to pixels in a one-thread-per-tile approach. This approach is detailed further in Chapter 3. An MPI approach is applied for distributed memory parallelism, in which each task retains a portion of the data which is rendered independently using the shared memory approach.

Composite: In the MPI parallel mode, task-dependent images must be composited to form the final image, which is performed via an MPI reduction operation.

Output: Finally, the resulted image must be written to disk. The Truevision TGA image format is used for encoding simplicity, and an image is written for each scene configuration defined in the parameter file, allowing the generation of offline movies.

The parallelism in Splotch is illustrated by the yellow boxes shown in Figure 2.6. Shared memory parallelism (on-node) is achieved through OpenMP and CUDA, applied to the projection, colouring, and rendering stages. Distributed memory parallelism is achieved via the Message Passing Interface (MPI), a standard library for

Single-Program-Multiple-Data (SPMD) parallelism. Further details on the Splotch code are available in the original and follow-up papers (Dolag, Reinecke, et al., 2008; Jin et al., 2010; Rivi et al., 2014), and also throughout the thesis. The remainder of Section 2.3 will detail the current optical model of Splotch and present an improved model to better support physical optical parameters and multi-wavelength data.

2.3.2 The Splotch Optical Model

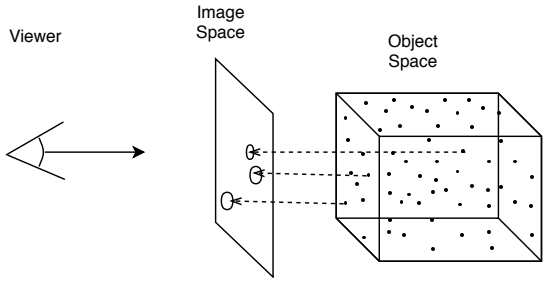


FIGURE 2.7: Volume splatting for particle data: particles are projected from object space to image space, and *splatted* across the image using a footprint function, typically a Gaussian kernel.

As introduced in Section 2.2.4, volume splatting is a well known object-order rendering algorithm. First each data element is transformed relative to a viewpoint, a parallel or perspective projection applied, and then the contribution of each element to line-of-sight rays cast from image pixels is computed using a 'splatting' kernel, summarised in Figure 2.7. In Splotch, each data element is represented by a particle, consisting of: a position in space (3D Cartesian coordinates), a 3D colour component, a radius value defining the size of the particle in 3D space, an intensity value that acts as a multiplier for the colour component, and a type field that allows particles to be rendered with type-specific parameters; the data structure used to store Splotch particles is shown in Figure 2.8. A simplified emission and absorption optical model is used to define each particles contribution to the rays. This optical model is illustrated in a different form to the general definitions in Section 2.2.1, to better describe a per-particle contribution and match the presentation for Splotch in Dolag, Reinecke, et al. (2008). Starting from the radiative transfer equation in differential form:

$$\frac{dI(x)}{dx} = (E_p - A_p I(x))\rho_p(x) \tag{2.7}$$

x is the coordinate along the line of sight, I is the intensity at position x, E_p and A_p are the emission and absorption coefficients of particle p. In this form, both E_p and A_p directly rely on $\rho_p(x)$, which defines the contribution to matter density interpolated from particle p, and is defined using a Gaussian distribution:

$$\rho_p(x) = \rho_{0,p} \exp(- \mid\mid x - x_p \mid\mid^2 / \sigma_p^2) \tag{2.8}$$

with x_p representing the particle coordinates, and $\rho_{0,p}$ and σ_p being the mass density and the radius of the particle respectively. For a more convenient compact support, the distribution is truncated at $\chi \cdot \sigma_p$, where χ is a suitably defined factor typically chosen such that $\chi \cdot \sigma_p \approx h$; where h is the intrinsic *smoothing length* of the particles (e.g. as described by Cossins (2010)). Due to this relation, the radius is commonly referred to as the smoothing length in this thesis and the referenced Splotch publications. As noted by Dolag, Reinecke, et al. (2008), the B_2-Spline typical for SPH particles is very similar in shape to the Gaussian distribution used here.

Following on from Equation 2.7, as shown in Appendix A.1, the contribution of a single particle to a ray is defined as:

$$I_{after} = (I_{before} - E_p / A_p) \exp(-A_p \int_{-\infty}^{\infty} \rho_p(x)\, dx) + E_p / A_p \tag{2.9}$$

with the integral provided by Equation 2.8.

For brevity, the frequency dependency of the intensity is not included in Equations 2.7 to 2.9; however, such a dependency does exist. The transfer function of Splotch supports independently treated emission in three frequencies (corresponding to R, G and B), but uses a simplified absorption coefficient, defined as a function of the emission coefficient like so:

$$A_p = A_c \cdot E_p \tag{2.10}$$

where A_c is a frequency independent constant absorptive factor. Furthermore, each particle has an intrinsic *type* property, used to distinguish particle species (for example gas, stars, and black holes). The transfer function can additionally be customised per type, to support an arbitrarily large range of functions each with three frequency outputs.

When rendering particles using the simplified emission and absorption optical model in Splotch, each particle induces a mixture of absorption and emission of energy in the viewing ray. This optical model requires the particle data to be sorted back-to-front with respect to the viewer, such that absorption and emission can be integrated in the correct order along the viewing ray, relative to the viewer. Out-of-order rendering would result in visual artifacts, such as a lack of (or inconsistent) occlusion from absorptive objects between the viewer and an emitting object. An emission only optical model can be solved in a commutative manner and so does not have this requirement. Splotch supports a further simplification for the emission and absorption optical model; an assumption that $E_p = A_P$ ($A_c = 1$) removes the need for order dependent rendering, and can be worthwhile approximation for highly diffuse or optically thin material, or extremely compact and bright material, both of which are common in astrophysical simulation (for example the intergalactic medium, and stars, respectively). As such, a flag can be set to indicate $E_p = A_P$,

neglecting sorting before rendering. This approach is utilised for MPI parallel distributed memory Splotch rendering, which will be addressed further in Chapter 3.

2.3.3 An Extended Optical Model

The existing optical model allows a pseudo-absorption, which is useful for e.g. providing 3D impressions in an orthographic projection as described in (Dolag, Reinecke, et al., 2008). However, fully independent absorption and emission coefficients are not implemented and so Splotch cannot support, for example, frequency dependent absorptive material. There are a variety of astronomical scenarios in which this is can be useful, as highlighted in Section 2.1.

The model can be extended by adding an independent absorptive coefficient, such that A_p in Equations 2.7 to 2.9 is defined as a frequency dependent physical quantity relating to particle p. To support this independent coefficient, an algorithmic extension is needed to account for the lower limit of absorption, i.e. for a purely emitting particle or one with negligible absorption, $A \approx 0$. In this case, to avoid floating point exceptions, equation 2.9 is replaced with:

$$I_{after} = I_{before} + E_p \int_{-\infty}^{\infty} \rho(x)\, dx \tag{2.11}$$

The Splotch particle structure must then be updated, requiring three additional fields to hold a per-particle absorption coefficient in three frequencies (RGB). The original particle structure for Splotch is defined in Figure 2.8 (left), requiring 36 bytes per particle. The updated structure is illustrated in Figure 2.8 (right), and requires 48 bytes per particle.

```
struct particle {
  float x
  float y
  float z
  float er
  float eg
  float eb
  float smoothing
  short intensity
  bool type
}
```

```
struct particle {
  float x
  float y
  float z
  float er
  float eg
  float eb
  float ar
  float ag
  float ab
  float smoothing
  short intensity
  bool type
}
```

FIGURE 2.8: The original structure defining a Splotch particle (left), using 36 bytes, and the new structure supporting a three-frequency per-particle absorption coefficient (right), using 48 bytes.

Finally an additional transfer function must be employed to convert A_p to $[R_p, G_p, B_p]$. As a first step, a similar function is implemented to that for E_p; which can be a transference of three data quantities, or a scalar colour table lookup, and

can be further varied based on particle characteristics. This function is implemented by particle *type*, allowing a variety of particle species to each be assigned different transfer functions as illustrated in the next section.

The additional fields and extended rendering algorithm outlined in this section are complimented with C pre-processor commands allowing them to be switched off at compile time. This supports a quick reversion to the simplified model described in Section 2.3.1, in the case that the reduced memory footprint is of greater importance than the image quality, or there is no sensible means of modelling the absorption.

2.3.4 Discussion

With these modifications, Splotch is able to support fully independent absorption, including self-absorption. However the physical model is still limited in comparison to those available in wider computer graphics literature. Particularly, the processes of scattering and illumination is currently disregarded, but form an intrinsic part of classical rendering (e.g. in the rendering equation (Kajiya, 1986)). Max and Chen (2010) present a follow-on to the seminal survey of optical models discussed in Section 2.2.1 (Max, 1995), describing in detail the physics of local and global illumination as applied to volume rendering. This is complimented by the comprehensive survey of illumination techniques for interactive volume rendering presented by Jönsson et al. (n.d.). One of the main reasons for avoiding such techniques is that, for astronomical simulation data, the number of light sources is typically extremely high. Beyond simple ambient lighting, it is simply not feasible to compute lighting for point sources that number in the hundreds of millions (e.g. stars in an galactic simulation). Having said that, there are use cases where modelling the effect of a small number of light sources can be useful, as seen in the work of (Magnor, Hildebrand, et al., 2005; Ralf Kaehler, 2013); further discussion in this vein is found in Section 2.5. Moreover, constant scattering affects could also be considered, such as the Rayleigh scattering applied by Kaehler et al. (2006).

In terms of the volume splatting paradigm, a common problem is the introduction of colour bleeding due to the non-piecewise integration of the splat reconstruction kernels. A solution for this is to organise splats into volume slices parallel to the volume face most parallel to the view, which are then composited. However this can introduce popping or flashing visual artefacts when sheets suddenly realign as the view changes. Such issues are discussed in detail by Mueller and Crawfis (1998), who also introduce an image-aligned slicing method to combat the artefacts of the volume aligned method. In Splotch, this is mostly a negligible issue as the contribution of a single particle is typically very minimal, meaning the approximate integration does not have a visible effect on the image. Particles are not used to represent surfaces, and so the problems inherent in reconstructing edges are not as important here as for other splatting applications, however the difficulty in representing sharp changes in field should still be investigated further. From the transfer function perspective, it would also be beneficial to support more complex functions

to produce multi-frequency emission on a per-particle basis, as opposed to the per-particle-species basis detailed in Section 2.3.3.

Finally, whilst there is still room for improvements, the current extended optical model now allows frequency-dependent absorptive media to be included in particle-based renderings, as demonstrated in the following section.

2.4 3D Visualisation of Observed Galaxies

The study of galaxy formation and evolution is a wide research field in astronomy, from galactic astronomy studying the Milky Way through observation, to intergalactic dynamics studies using large N-body computational simulations. Observers try to understand the phenomenology of galaxies through detailed observational studies, using many varieties of image based analysis techniques to expand our wider understanding of the many physical processes leading to the formation and evolution of galaxies.

Those galaxies actively visible in the sky can typically only be observed from a single viewpoint, that is, from the point of view of an Earth-based observer. As such, astronomers aim to observe as many as possible through large sky surveys, capturing and categorising huge variety of galaxies in different shapes, sizes, and at different angles to our single observing point (usually defined by the *galaxy inclination* and *position angle*). To illustrate this, Figure 2.9 shows two extreme-cases, the face on view of spiral barred galaxy M83, and the edge-on view of spiral galaxy NGC4565.

FIGURE 2.9: The extreme ends of spiral galaxy inclinations: M83 (*left*) shows the full complexity of its spiral structure face-on, with dark reddish brown dust features. The edge-on NGC4565 (*right*) presents a clear bulge and strongly obscuring dust lanes in the galactic plane. *Left* and *right* images courtesy of *NASA, ESA, and the Hubble Heritage Team (STScI/AURA)* and *Robert Franke*[2] respectively.

[2] http://bf-astro.com/index.htm

Whilst such galaxies can only be observed from a single viewpoint, the variety of imaging data collected via telescopic instrumentation, and the derived knowledge based on such data, can allow a partial reconstruction of the morphology and kinematics of such galaxies. In this section, the visualisation methodology of previous section will be utilised to generate 3D representations of recognisable galaxies, highlighting the different material components and their physical characteristics. To that end, the following subsections describe a novel pipeline for generating and visualising 3D models of galaxies using observed and inferred data, enabling the reconstruction and 3D exploration of individual, or groups of, galaxies. Splotch is exploited for 3D visualisation, using the improved optical model described in Section 2.3 to enable the addition of a dust component, exploiting the component-dependent transfer function to distinguish the various emissive and absorptive galactic components.

Whilst reconstruction of astronomical phenomena has been an active research topic in astronomy for many years, typically the focus has been on quantitative understandings of morphology; conversely, reconstruction of astronomical phenomena based on observational data for realistic visualisation has not been deeply explored in the literature. Nadeau et al. (2001) study the visualisation of stars and emission nebula, using a 3D modelling technique for the Orion nebula developed in (Wen and O'dell, 1995). The work of Magnor, Kindlmann, et al. (2004) and Magnor, Hildebrand, et al. (2005), also referenced in previous sections, presents a comprehensive modelling and visualisation pipeline for planetary nebulae, based on a novel *Constrained Inverse Volume Rendering (CIVR)* technique to reconstruct gas distributions from a series of optical images. The CIVR technique further underpins the work of (Hildebrand, Magnor, and Fröhlich, 2006), who reconstruct and visualise spiral galaxy M81 based on optical images. Further discussion comparing the presented work to such existing literature can be found in Section 2.4.6.

As a case study to demonstrate the pipeline, this work focuses on the nearby barred spiral galaxy known as M83 (shown in Figure 2.9 (*left*)), for its designation in the Messier catalogue, or as NGC5236 in the New General Catalogue of Nebulae and Clusters of Stars (NGC). As one of the closest and brightest barred spiral galaxies in the sky, M83 has a large collection of multi-wavelength survey data available making it an ideal case study for this work.

2.4.1 The Modelling Pipeline

Figure 2.10 shows the key stages of the galaxy modelling pipeline, illustrating the various inputs and outputs. The following sections provide a brief overview of each stage of the current pipeline and the visualisation process for M83.

2.4.2 Observational Source Data

As introduced in Section 2.1, astronomical phenomena emit energy not only as visible light but as radiation across the whole range of the EM spectrum. Indeed, the

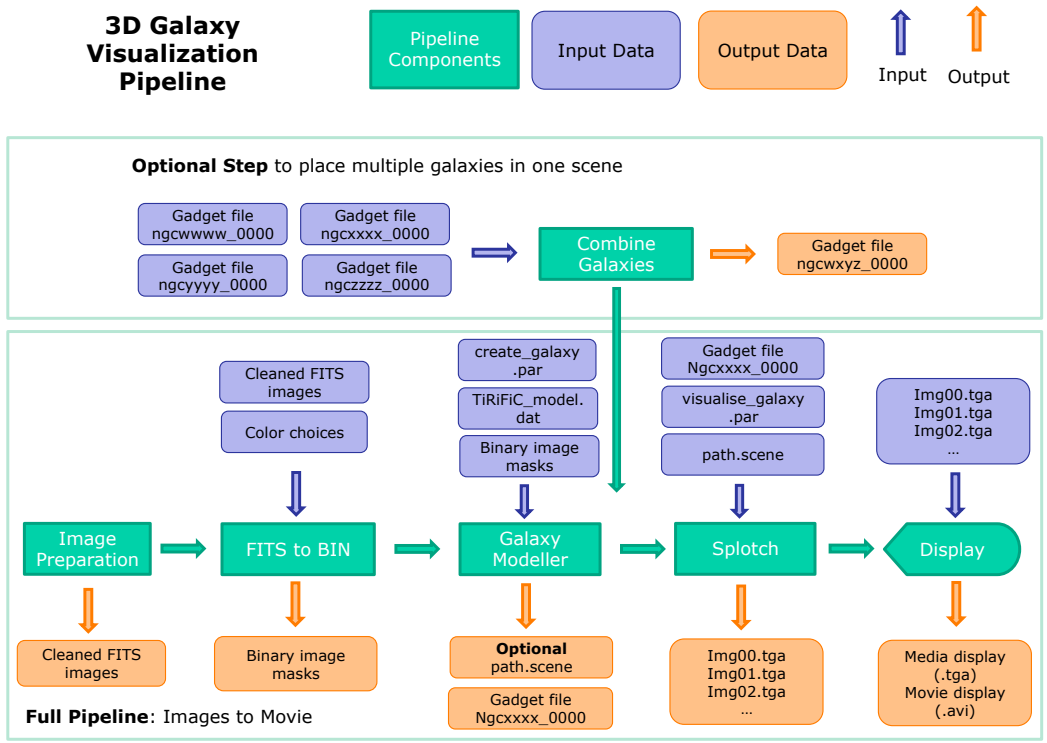


FIGURE 2.10: A stage-by-stage representation of the galaxy modelling pipeline. Each of the stages, inputs, and outputs are described further in the text.

various components of a galaxy may emit radiation across a large range of the spectrum, meaning that a comprehensive understanding of an object requires observation across multiple wavelengths. For example, bright stars are typically captured in the optical and ultraviolet (UV) wavelength whilst cold atomic hydrogen gas (H I), which typically shows structure extending far beyond the bright stellar disk, must be observed in the radio wavelength. In the current pipeline, the first step is image preparation. Observed images are first collected in the optical, UV, infrared (IR) and radio wavelengths; the specific survey data used depends on that which is available for a particular galaxy.

For M83, optical *Ha*- and *R*-band images are used from the Survey for Ionization in Neutral Gas Galaxies (SINGG) (Meurer et al., 2006) and near- and far-*UV* images from the Galaxy Evolution Explorer (GALEX) (Gil de Paz et al., 2007) to inform the stellar components. The dust distribution, which is to varying degrees already part of the observed stellar component, is informed by 8μm images from the Spitzer Infrared Array Camera (IRAC) instrument (Dale et al., 2009). IR emission in the 8μm is typically used to identify and study interstellar dust, as an example see the dust maps built by Helou et al. (2004).

To represent the structure of the gas component, radio interferometric H I intensity maps at up to three different resolutions are collected from the Australia Telescope Compact Array (ATCA) data. Most importantly, the H I data is used to determine the shape and kinematic properties of galaxies, described in the next section.

The "Local Volume H I Survey" (Koribalski et al., 2018), provides ATCA H I data for nearly 100 nearby galaxies ($D < 10$ Megaparsec) which are publicly available[3].

The images are then pre-processed and organised as a set of FITS (Pence et al., 2010) files. Pre-processing is a difficult and manual task requiring cleaning the images of fore- and background sources, selection of intensity clipping ranges (due to high dynamic range of pixel values), then orienting and scaling the images in a uniform manner. This process is carried out by an experienced astronomer. Figures 2.11 and 2.12 show the collection of images used for M83 after pre-processing.

The cleaned FITS images are then input to the *FITS to BIN* pipeline stage, consisting of a preprocessing script that extracts, clips, and colours the FITS image pixels. Colours are chosen typically to highlight the different components, and if possible to match existing composite observed images. Binary intermediate files contain per-image pixel maps that are used as direct inputs to the galaxy modeller.

2.4.3 Kinematic modelling of Warped Disks

After gathering a collection of observed data on a specific galaxy, the next step is to reconstruct the 3D structure. A best-fit model is used to describe the galaxy kinematics, and particularly the warped outer disk typical of spiral galaxies as can be seen in the observed images of M83 in Figure 2.12. H I data cubes and/or derived velocity fields are used to obtain the best fitting shape and kinematic properties; the galaxy inclination (i), position angle (PA), and rotational velocity (v_{rot}), as a function of radius. To ensure reasonable results, these fits are made as part of a tilted ring analysis, a long-standing and successful approach to investigating kinematic galaxy structure (Rogstad, Lockhart, and Wright, 1974).

For M83, a tilted ring model generated with the TiRiFiC software (Józsa et al., 2007), provides a set of concentric ellipses of varying radii, with i, PA, and v_{rot} values as shown in Table 2.1. This model, visualised in Figure 2.13, effectively describes the warped disk structure of M83, and is used as the basis of the 3D model generated in the next section, represented as the *TiRiFiC_model.dat* input to the Galaxy Modeller stage in Figure 2.10.

2.4.4 Building the 3D Model

The Galaxy Modeller uses a particle based approach to model a series of separate galactic components, each of which is generated based on available observed and/or derived data, and tuned experimentally using additional variables. The components currently present in the model are:

- Stellar population
- Diffuse gas

[3]LVHIS page: www.atnf.csiro.au/research/LVHIS

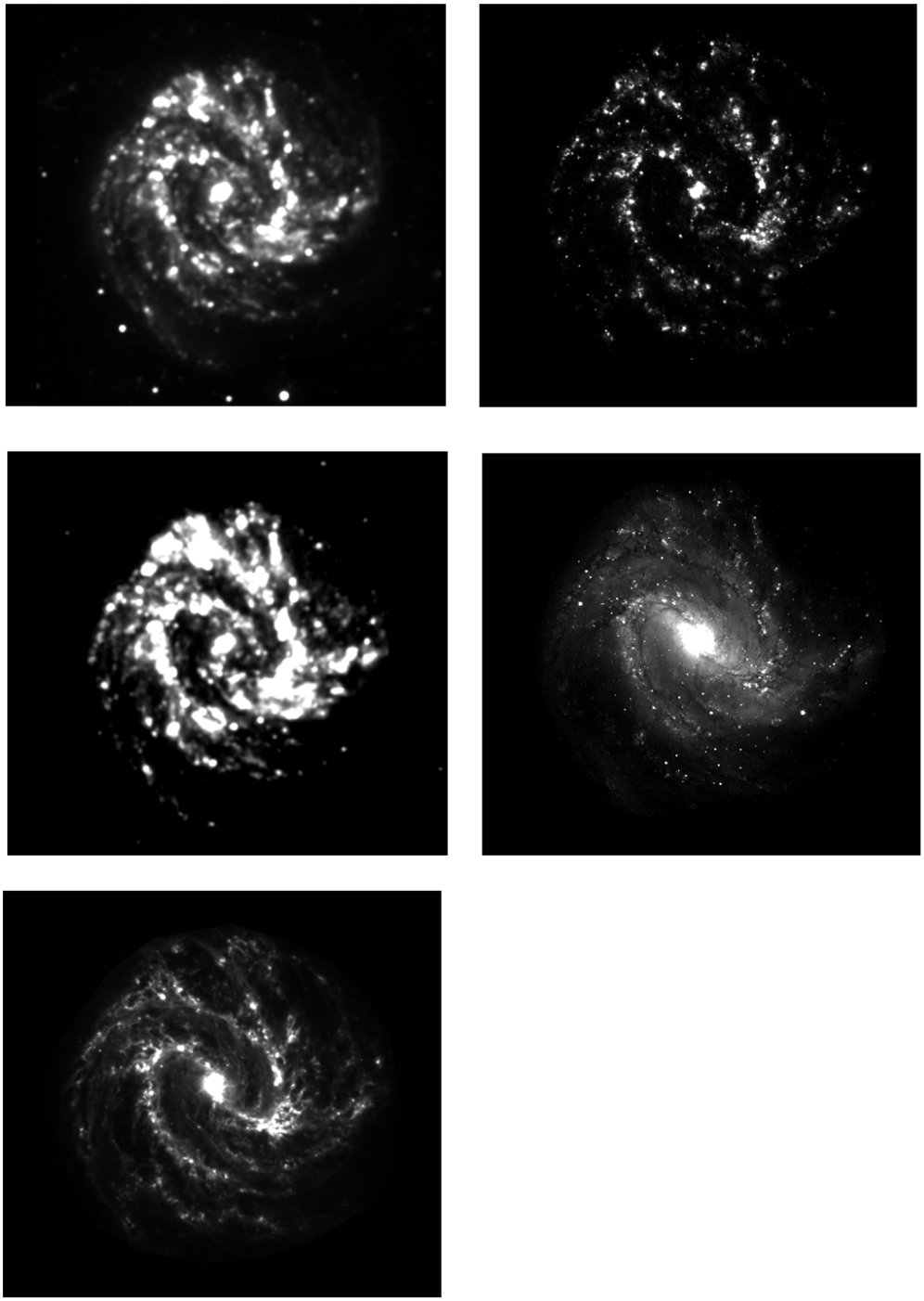


FIGURE 2.11: The cleaned FITS images of M83's inner disk in ultraviolet, optical, and infrared bands. *Top left:* GALEX near UV-band. *Middle left:* GALEX far UV-band. *Bottom left:* SPITZER IRAC 8mu-band. *Top right:* SINGG *Ha*-band. *Middle right:* SINGG *R*-band.

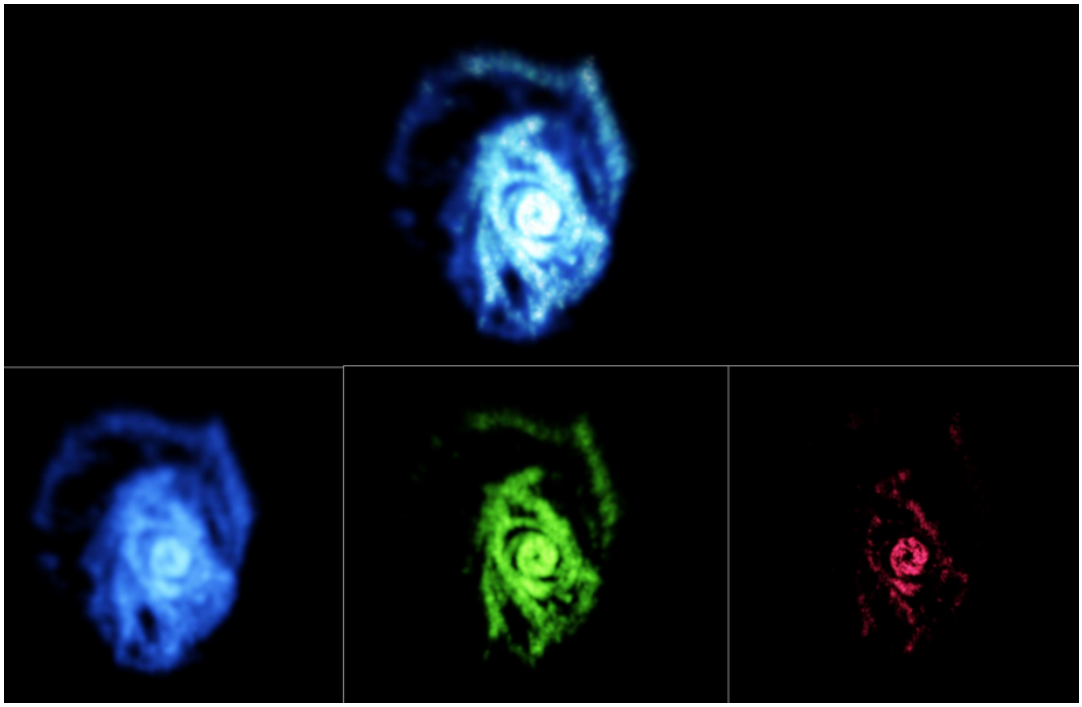


FIGURE 2.12: M83: cleaned FITS images of the warped outer disk in high, medium, and low spatial resolutions. The differing resolutions are defined by a weighting applied when extracting images from the raw data, allowing to tune the extent of detected emission. *Bottom left:* Low. *Bottom Middle:* Medium. *Bottom Right:* High. *Top middle:* Combined resolution. Figure and data courtesy of Baerbel Koribalski, Australia Telescope National Facility, CSIRO.

RADIUS	VROT	Z0	INCL	PA
0.00E+00	1.50E+02	2.99E+01	2.37E+01	2.27E+02
1.72E+02	1.96E+02	2.99E+01	2.37E+01	2.27E+02
3.43E+02	2.00E+02	2.99E+01	2.47E+01	2.26E+02
5.15E+02	1.95E+02	2.99E+01	3.42E+01	2.19E+02
6.86E+02	1.92E+02	2.99E+01	4.48E+01	2.09E+02
8.58E+02	1.90E+02	2.99E+01	5.06E+01	2.01E+02
1.03E+03	1.89E+02	2.99E+01	5.10E+01	1.96E+02
1.20E+03	1.89E+02	2.99E+01	5.08E+01	1.92E+02
1.37E+03	1.89E+02	2.99E+01	5.01E+01	1.88E+02
1.54E+03	1.89E+02	2.99E+01	4.89E+01	1.88E+02
1.72E+03	1.90E+02	2.99E+01	4.33E+01	1.88E+02
1.89E+03	1.89E+02	2.99E+01	3.69E+01	1.82E+02
2.06E+03	1.89E+02	2.99E+01	3.06E+01	1.74E+02
2.23E+03	1.89E+02	2.99E+01	2.49E+01	1.65E+02
5.23E+03	1.89E+02	2.99E+01	2.49E+01	1.65E+02

TABLE 2.1: A minimal list of tilted ring model values for M83, visually represented in Figure 2.13. For brevity only 15 representative rings are shown, the full model contains ∼300 rings and was generated by Peter Kamphuis using the TiRiFiC tilted ring fitting software (Józsa et al., 2007).

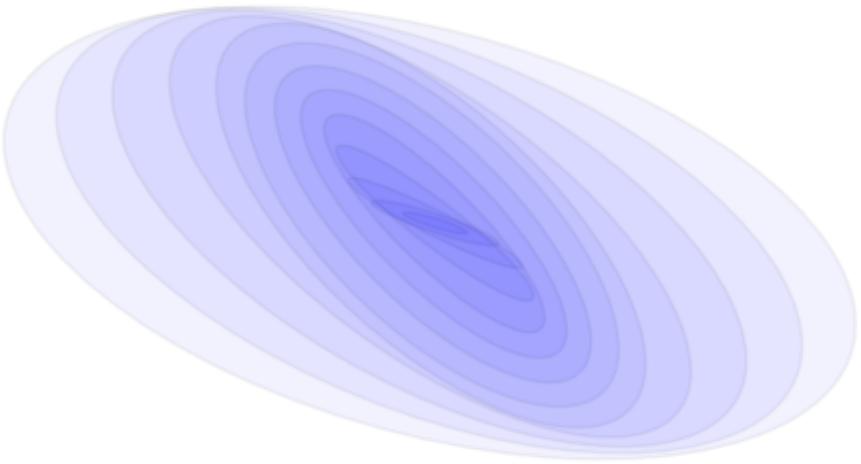


FIGURE 2.13: A visual illustration of the TiRiFiC tilted ring model for M83, based on the data in Table 2.1.

- Dust
- Galactic bulge
- Globular clusters surrounding the galaxy
- Diffuse stellar, or dark matter, halo

A particle based distribution method is used to model the stellar, diffuse gas, and dust particle populations of the disk. In the following description, coordinates (x, y, z) are used relative to the galactic plane, such that x, y are in-plane and z is axial. Firstly, a 3 dimensional distribution of N_p *seed* particles is generated randomly on the galactic plane at $z = 0$, following the concentric tilted ring kinematic model described in Section 2.4.3. This distribution of particles in object space is projected onto the image plane of one of the observational images, Obs_λ. Each seed particle P is assigned an intensity I_s as a function of the intensity of the image pixel I_p with which it intersects, along with a scalar or three-component RGB_s colour value defined by the binary image masks input to the modeller (in Figure 2.10). An intensity threshold is defined as I_t, defaulting to 0, and all seed particles with $I_s < I_t$, are removed, i.e. those that lay in an area of the disk with no emission in wavelength λ.

The disk thickness is given by point clouds generated around the remaining seed particles. For each of the remaining particles, a point cloud of size N_{star} is generated according to a Gaussian distribution defined by σ_{star}, and N_{star}. Each spawned particle inherits the intensity I_s and colour values RGB_s of the seed particle, and is further characterised by a *smoothing length* h, estimated as the average inter-particle distance. N_{star} and σ_{star} define the number and displacement of particles from the galactic plane, and can be calculated using one of three available models:

1. A constant Gaussian thickness, where the Gaussian displacement of particles from the galactic plane in each dimension is defined by an input parameter σ_{xyz} that is constant across dimensions. Particles are distributed with $\sigma_{star} = \sigma_{xyz}$ and N_{star} is scaled with I_s. As such, brighter pixels of the image are represented by more points across a larger spatial volume.

2. Radial (H_r) and axial (H_z) scale heights are provided, along with input parameter σ_{xy}. N_{star} is scaled proportionally to $exp(-R/H_z)$ (where R is radial distance of the particle to the center of the disk), and the particles are distributed with σ_{xy} in the radial plane and σ_z axially, allowing a tapered disk structure to be represented

3. Thickness is scaled to fit to known measurements of flared disk thickness in edge-on galaxies, following the measurement of $FWHM_{z,g}$ (gas layer thickness) as presented in Figure 25 of O'Brien, Freeman, and van der Kruit (2010).

The scaling of N_{star} with I_s is defined as the inverse of the function defining I_s from I_p, such that $N_{star} \cdot I_s \approx I_p$.

As can be seen in high quality edge-views of spiral galaxies, for example that shown in Figure 2.14, dust lanes can have complex morphologies with cloud and filament-like structures, and can be very well-defined against the bright stellar background. To reflect this, the thickness of the dust component is implemented using a different scheme. The point clouds generated around seed dust particles are distributed using one of two models:

1. Dust filaments are approximated via assigning velocities to particles randomly distributed around the linear velocity as defined by the TiRiFiC model. A random walk is traced, generating a new particle at each step, with a gravitational factor applied such that filaments are constrained near to the galactic plane.

2. As with the previous item, however particle velocities are distributed around the rotational velocity of the disk.

Dust is discussed further in Section 2.4.5.

The galactic bulge is more simply defined as a Gaussian distribution of points defined by input parameters $sigma_x$, $sigma_y$, $sigma_z$, and N_{bulge}, placed in the centre of the galaxy in line with typical observations, and coloured white. The bulge can be rotated to conform to the tilted-ring kinematic model. The globular cluster, and diffuse stellar/dark matter halo components are available in the pipeline for visual effect in a multi-galaxy scene, and not used in this work.

Each of the components is generated sequentially, and the final model is composed and written as a CSV, Gadget, or HDF5 file. All of the tunable parameters can be provided though an input key-value text file shown as *create_galaxy.par* in Figure

[4] http://www.caelumobservatory.com/gallery/n891_32in.shtml

FIGURE 2.14: An excellent demonstrative example of the complex morphology that can be found in galactic dust lanes, NGC891 is a large spiral galaxy that is almost exactly edge-on as viewed from Earth. Image courtesy of Adam Block/Mount Lemmon SkyCenter/University of Arizona[4].

2.10. The binary image masks that provide RGB_s colour values are manually chosen to visually distinguish each of the galactic components, and if possible to resemble existing recognisable false-colour composite images.

2.4.5 Visualisation

Splotch is used for the visualisation stage of the pipeline, and takes as an input: the data file written by the Galaxy Modeller (e.g. *NgcXXXX_0000* as shown in Figure 2.10), a key-value parameter file (*visualise_galaxy.par*) describing the scene configuration, and optionally a scene file (*path.scene*) which can be used to describe a set of scene configurations for an animation.

For each of the galactic components, which are treated as separate particle species (as introduced in Section 2.3.2), a series of tunable visual parameters are available. The smoothing length h of particles can be scaled using a *size* parameter, and the intensity I can also be scaled using a *brightness* parameter.

The emission and absorption coefficients of each component are defined by the intensity I of the particle, retained as I_s from the modelling stage, such that the emissivity of the particles is directly related to the observed intensities of the galactic component to which the particle belongs. The emission coefficient in each frequency of the final image E_{RGB} is defined as $I_s \cdot RGB_s$. The absorption coefficient A_{RGB} is defined as $I_s \cdot RGB_A$, where RGB_A is a three component absorption profile provided via lookup table during transfer function evaluation.

In the example case for M83, there are 5 galactic components included in the visualisation, which are informed by observations as illustrated by Table 2.2. The stellar distribution and diffuse gas are treated as coloured emissive sources, where $E_{RGB} = I_s \cdot RGB_s$ and $A_{RGB} = 0$. The bulge component is treated as a white emissive

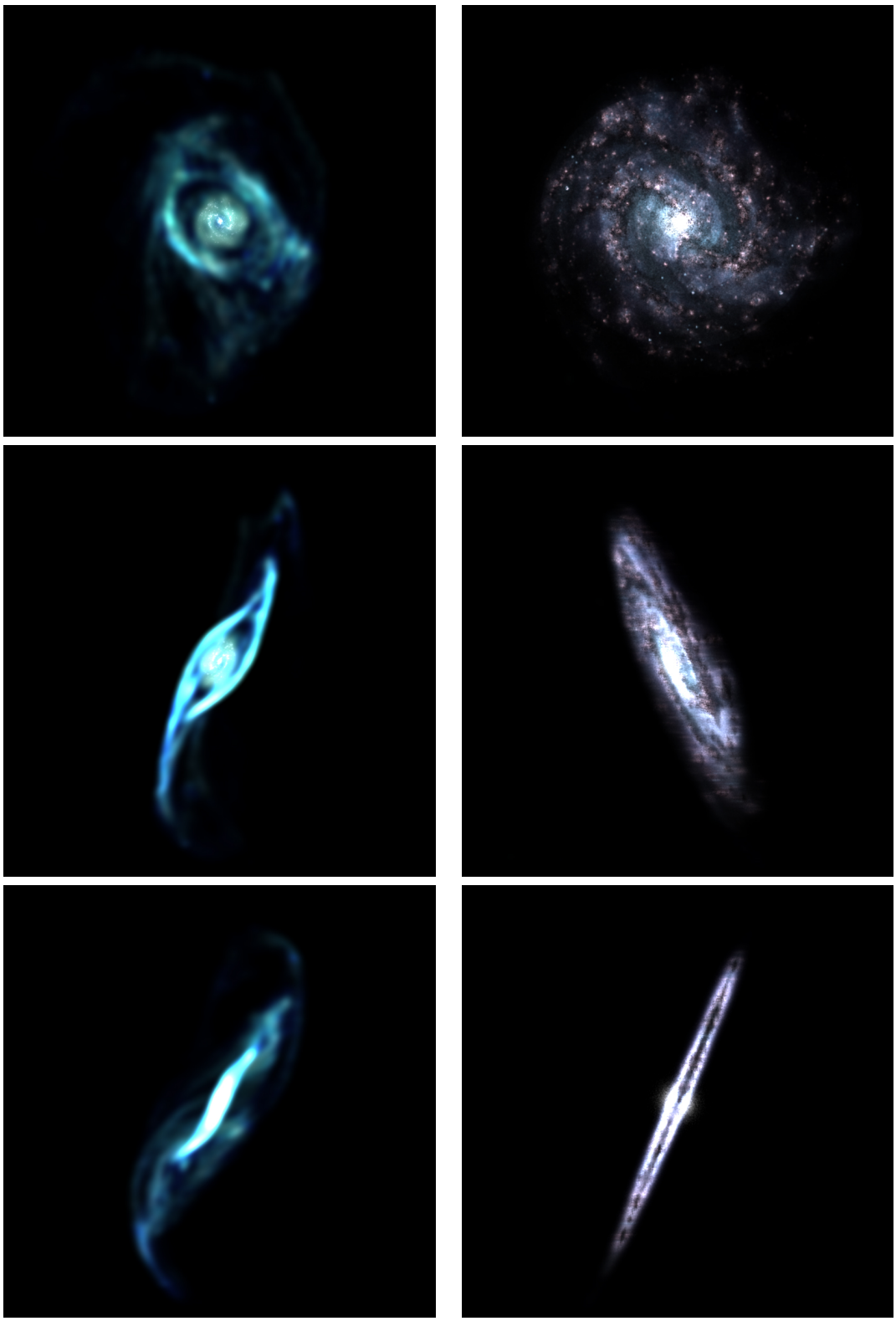


FIGURE 2.15: The results of applying the visualisation pipeline to M83, split into *close* and *far* (*right* and *left* respectively), showing face-on, angled, and edge-on views (*top* to *bottom* respectively). The far images include the H I extended gaseous disk (∼100 kpc max radius), showing clearly the large warped structure. The close images have the H I removed to more closely resemble an optical composition (∼13 kpc max radius), and show clearly the absorptive dust lanes, tapered disk and stellar bulge.

Galactic Component	Source Imaging Data
Stellar distribution (Optical)	Optical *Ha*- and *R*-band SINGG
Stellar distribution (UV)	UV Near and Far GALEX
Diffuse Gas	HI 3 Resolution ATCA
Dust	IR 8um Spitzer IRAC
Galactic Bulge	None

TABLE 2.2: A mapping demonstrating the relationship of galactic components used for M83 to source image data.

source, where $E_{RGB} = I_s \cdot [1,1,1]$ and $A_{RGB} = 0$. The dust is defined as a grey absorptive component, where $RGB_s = [0,0,0]$ and $A_{RGB} = I_s \cdot [1,1,1]$.

Figure 2.15 demonstrates the results of applying the visualisation pipeline to M83, split into *close* and *far* (*left* and *right* respectively), showing face-on, angled, and edge-on views (*top* to *bottom* respectively). The far images include the HI extended gaseous disk, showing clearly the large warped structure, which is not easily discernible from the observed images (Figure 2.12). The maximum extent of the HI region is ∼100 kilo-parsecs (kpc) (Koribalski et al., 2018). The close images show the inner disk (∼13 kpc diameter (Thilker et al., 2005)) and have the HI removed to more closely resemble an optical composition, and show clearly the absorptive dust lanes, tapered disk and stellar bulge.

2.4.6 Discussion and Evaluation

To the best of the author's knowledge, this is the first such combined modelling and visualisation pipeline for galaxies to target a general group of galaxy categories using a wide array of observational input sources, particularly including multi-resolution radio data. Previous works have covered a wide range of frequency bands, such as (Li, Fu, and Hanson, 2008), but are focused on exploration of spectral cubes, and others have modelled spiral galaxies using different modelling and visualisation techniques exploiting fewer observational sources (Hildebrand, Magnor, and Fröhlich, 2006).

From a modelling and visualisation perspective, this approach is focused on enhanced physical realism in galaxy models. It is infeasible to actually observe specific galaxies from another angle, and as such impossible to be certain of physical accuracy; the aim of this work is to physically motivate the galaxy representation using state of the art data from observation complemented by analytical and numerical modelling. To evaluate the extent to which this is achieved, a series of high level morphology and visual properties have been identified to comprehensively describe a galaxy from a modelling point of view, and the methods used to determine these properties for each galaxy component have been categorised by their physical basis. The properties, with reference to the previous sections, are:

Radial seed distribution This property defines the distribution of seed particles (as discussed in Section 2.4.4) in the radial plane of the galaxy (i.e. distribution across the galaxy face).

Axial seed distribution This property defines the distribution of seed particles in the axial plane of the galaxy (i.e. galaxy thickness).

Particle size and distribution This property defines the size of each representative particle and distribution of such particles around the seed (i.e. galaxy density).

Emission This property defines the value used as the emission coefficient (as discussed in Section 2.4.5) for each particle during visualisation (i.e. galaxy brightness and colour).

Absorption This property defines the value used as the absorption coefficient for each particle during visualisation (ie. galaxy absorption, dust lanes, and shadows).

The method or model used to define each of these properties per component has been categorised into one of the following groups, ordered from first to last by the strength of the physical basis for the method:

Source data The property has been derived directly from observational data of the source object being modelled.

Observed model The property has been derived from general observations or relations observed for the object, or type of object, being modelled.

Theoretical model The property has been derived from a theoretical model or simulation of the object, or type of object, being modelled.

Scaled to source data The property has been scaled according to source data, but may be initially defined by heuristic.

Heuristic model The property has been defined heuristically in collaboration with an experienced astronomer based on resultant visual effects.

As illustrated in Table 2.3, whilst a good portion of the modelling and visualisation pipeline is strongly physically motivated, there are still a series of methods based on heuristics. In particular, further work is envisioned to develop a more realistic axial dust model, considering the existing research into the morphology and scattering effects of dust grains in both in astronomy and visualisation research such as (Magnor, Hildebrand, et al., 2005), who use a 3D Perlin noise model (Perlin, 1985) to describe dust morphology combined with careful treatment of dust scattering. Furthermore, the axial distribution of the bulge component may be more robustly defined based on observed properties in systematic studies of bulges in galaxies, for example Gadotti (2009) and Fisher and Drory (2011).

From an astronomical perspective, the modelling and visualisation pipeline provides a means of dynamic, 3D, visual exploration of galaxies that are typically only seen in 2D. In general, a 3D representation can allow a deeper understanding of galactic morphology; for example distinguishing whether given features are real

Galaxy Component	Component Property				
	Radial seed distribution	**Axial seed distribution**	**Particle size and distribution**	**Emission**	**Absorption**
Bulge	Observed model	Heuristic model	Heuristic model	Heuristic model	None
Stellar	Source data	Observed model	Heuristic model	Scaled to source data	None
Dust	Source data	Heuristic model	Heuristic model	Scaled to source data	Scaled to source data
Diffuse gas	Source data	Theoretical model	Heuristic model	Scaled to source data	None

TABLE 2.3: A summary of the physical basis for the various configurable properties of each galaxy component.

characteristics or projection effects, or highlighting properties hidden by a 2D representation. Visualisation has a unique capacity to convey scientific content, helping scientists to explain complex concepts to colleagues, other professionals, and particularly non-experts such as the public, funding agencies, and decision makers. Utilising Figure 2.15 as an example, this work is able to clearly highlight the co-existence of extended H I outer disks (figure left) and inner stellar disks (figure right), only the latter of which is typically well known to the public.

A caveat of the modelling and visualisation pipeline is the lack of applicability to images of edge on galaxies. The further from face-on a galaxy is tilted, the more difficult it is to reconstruct in 3D, and edge-on galaxies simply do not contain enough information to realistically construct a face-on impression. One approach to address this problem from a visual perspective could be to generate a believable face-on impression through statistical analysis of the many existing observations of face-on galaxies, or exploiting recent results of high resolution cosmological simulations such as EAGLE (McAlpine et al., 2016) to realistically describe particle distributions in galaxies.

Moreover, as the physical realism of the modelling and visualisation increases, it may be possible to further use this pipeline to create mock observations. Such observations are synthetic images of theoretical data which can be used for the setup and tuning of observational instruments, along with image-based comparison between results of theoretical simulation and observation (e.g. (Bottrell et al., 2017; ZuHone et al., 2018)) for validation purposes.

Finally, beyond improving the visual quality and physical motivations of the models, future work would focus on introducing more automated pipeline stages, enabling new galaxies to be modelled faster with less manual intervention. This would include further development of the Combine Galaxies pipeline stage, which can be used to combine multiple galaxies into a single scene. This would form a necessary basis for visual fly-throughs of a local galactic neighbourhood, providing a unique way to explore groups of galaxies.

2.5 Beyond the Visualisation: Radiative Transfer in Astrophysics

Radiative transport methods are useful not only in computer graphics, but also to solve problems in the physical sciences. For example, many astrophysical processes are affected by radiation transport, relying on the same physical foundations as volume rendering in computer graphics (the basics of which are outlined in Section 2.2.1). This point is also highlighted by Peters (2014), who provides an introduction to volume rendering for astrophysicists. This theoretical overlap has led to a variety of existing examples of inter-disciplinary knowledge transfer between astronomy and computer graphics. For example, as noted by Max (1995), the P-N method for multiple scattering was first introduced by Chandrasekhar (1950) for stellar atmospheres, before later being applied in computer graphics for volume rendering (Kajiya and Von Herzen, 1984), and Flux Limited Diffusion (FLD) methods (Levermore and Pomraning, 1981) have also been explored in graphics, again for multiple scattering (Koerner et al., 2014).

Radiative transport codes used in astrophysics can also be effectively used for volume rendering, as demonstrated by Peters (2014) using the code RADMC-3D. One approach to better motivate the physical realism in Splotch for astronomical data, is by further exploring this crossover and better understanding radiation transport in astronomy. The work in this section presents a particle based radiative transport algorithm, STARRAD, developed for use within the context of astrophysical simulations. This is followed by a discussion of the potential improvements that could be made to Splotch based on the physical implementation of STARRAD (Section 2.5.5).

2.5.1 STARRAD: radiative transfer for SPH simulations

Astrophysics simulations are currently able to capture many of the most relevant hydrodynamical and gravitational processes involved in the formation and evolution of stars, planets, and galaxies. A popular simulation approach is based on Smoothed Particle Hydrodynamics (SPH), a Lagrangian numerical technique for computational fluid dynamics. SPH methods track fluid flow by discretising the computational domain into free-flowing particles, and can be effectively used to treat hydrodynamical processes in astrophysical systems. A thorough introduction to SPH methods can be found in (Monaghan, 1988).

The use of radiation transport in such simulations has been limited, despite its importance in a number of physical scenarios. There are two primary reasons for this:

- Short timescales

  The timescale upon which radiative processes must be followed is much shorter than that of hydrodynamical processes. This is because the speed of light (for radiation treatment) is roughly 10^4 times faster than the speed of

sound (for hydrodynamics) in the molecular clouds where stars are forming. As such, simulation timesteps for radiation must be much smaller than for hydrodynamics, which can incur great computational expense.

- Modelling complexity

  The processes of absorption, emission, and scattering are difficult to treat in the majority of intrinsically 3D astrophysical simulations, which have a large computational volume to be treated and potentially many radiative sources.

However, modern computing resources and approximate methods for radiative transport are motivating the inclusion of radiation transport in SPH simulations (as discussed in (Reed, Dykes, et al., 2018)), which is important for the dynamics of many astrophysical processes. Radiative feedback from stars and black holes plays a dominant role in regulating the rate of gas accretion into galaxies and star formation within galaxies. The accumulated regulation of star formation determines the observable galaxy properties such as stellar luminosity, colour, and age. Furthermore, the combination of multiple modes of radiation transport allow new classes of problems to be solved. As an example, which will be referred to in the remaining sections, a planet-forming disk around a protostar requires modelling the star light from the protostar as well as diffusion of that radiation outward through the planet forming disk. Evolving the disk temperature accurately, in this case, is needed to follow the fragmentation of the disk which ultimately determines the number and masses of planets that form.

The following subsections introduce STARRAD, a radiative transfer algorithm for modelling direct lighting of a gaseous medium around a source, developed within the context of the DIAPHANE project, a suite of radiation algorithms for astrophysical simulations. An overview of the algorithm implementation and validation is presented, also forming the authors contribution to the DIAPHANE project collaborative publication (Reed, Dykes, et al., 2018) which presents the full suite of algorithms.

2.5.2 STARRAD Algorithm

The STARRAD algorithm is based on a ray-casting approach designed to treat a small number of heating sources, such as the previously introduced example of modelling the direct radiative effect of a protostar on its surroundings STARRAD is based on a HEALPix (Górski et al., 2005) decomposition of the sphere of influence around the source. Figure 2.16 demonstrates the HEALPix approach, which is an effective means of discretising a sphere into equally sized pixels, through which rays can be cast from the source into the surrounding medium. This approach allows SPH data to be processed directly in particle form, rather than re-sampling to a regular grid.

For each pixel, a ray is allocated of length l, and further discretised into segments of length s. Three tunable parameters control the resolution of spherical grid: the density of the ray distribution (synonymous with the resolution of the HEALPix

distribution), the total length l of each ray which is chosen relative to the size of the computational domain, and the length s of each ray segment which is chosen relative to the intrinsic particle smoothing length h.

The algorithm is then structured in three key phases (Figure 2.17):

1. **Optical depth**: Sample the volume for optical depth contributions at discrete points along each ray, stored in ray segments. This is accomplished by iterating once through the particles, calculating particle-ray intersections using HEALPix intersection tests, and spreading the particle optical depth across the affected ray segments.

2. **Ray integration**: Starting from the source intensity, solve an absorption-only model of the radiative transport equation along each ray, resulting in a radiation intensity field surrounding the source.

3. **Heating**: Apply the radiation intensity field to heat the medium, by applying the same object-order iteration as in Step 1, and calculating the appropriate amount of energy absorption from each ray segment.

The optical depth of a particle in Stage 1 is defined, (as in Section 2.2.1, repeated here for context), by:

$$\tau(x0, x1) = \int_{x0}^{x1} \alpha(x)\, dx \tag{2.12}$$

where α is described as:

$$\alpha = \kappa\rho(x) \tag{2.13}$$

The absorption coefficient, κ, is defined by an appropriate lookup table (as discussed in Section 2.5.3), and Equation 2.12 is integrated from the entry of the ray to the particle radius of influence, to the exit.

Note to calculate heating in this case the dependency on frequency is not considered, as such the intensity I_0 of the source is defined starting from the Stefan-Boltzmann law (instead of the Planck law as described in Section 2.2.2):

$$j = \sigma T^4 \tag{2.14}$$

The radiant exitance j is converted to a per-ray initial intensity I_0 by multiplying by the source surface area and then dividing by the number of HEALPix rays.

Having calculated initial intensity I_0 and optical depth values for each ray segment, the rays are integrated via the typical *trapezoidal rule* (Springel, 2016) to generate a radiant intensity field around the source (Stage 2).

Finally, in Stage 3, an object-order iteration over the particles is repeated to calculate absorbed energy from the intensity field. Particle-ray intersections are calculated, and instead of contributing optical depth to the ray, the opacity is used

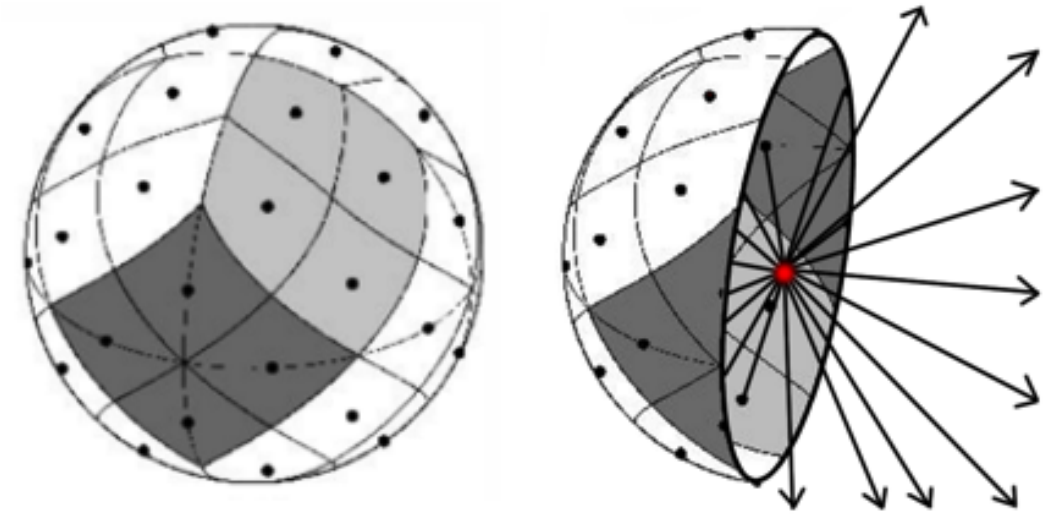


FIGURE 2.16: HEALPix decomposition of the sphere around a heating source, figure modified from (Górski et al., 2005).

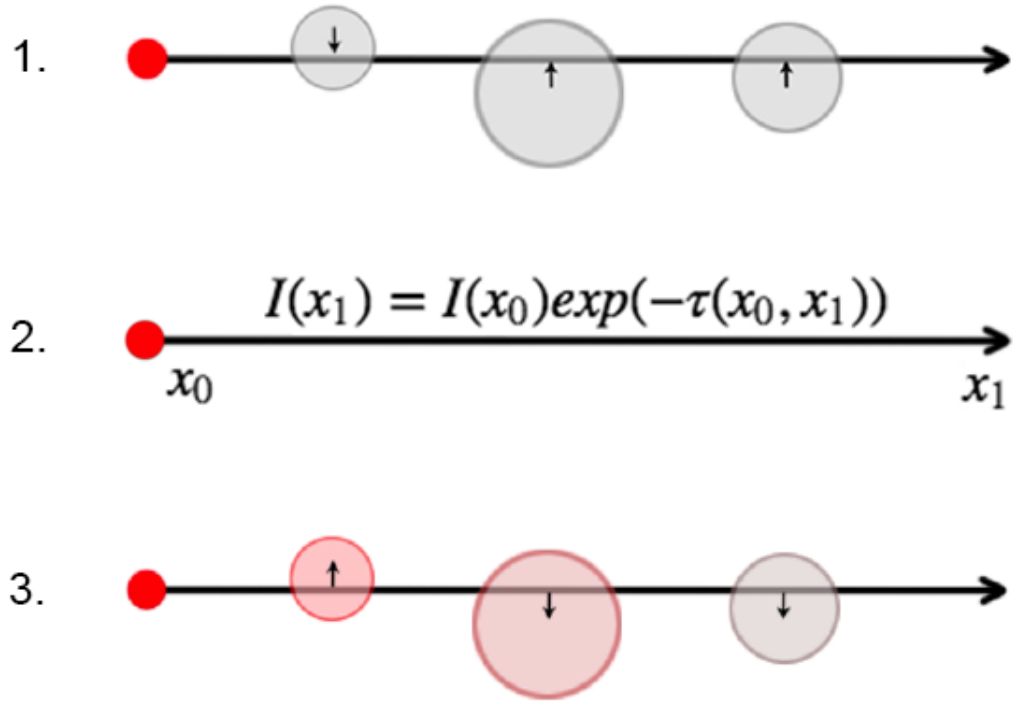


FIGURE 2.17: Three phases of the STARRAD algorithm; optical depth, ray integration using the absorption model, and heating, respectively.

to calculate energy absorption. The energy absorbed per gram U is calculated in $ergg^{-1}$, and then particle heating is determined determined following an ideal gas equivalence (Springel, 2016):

$$dT = \frac{UuM_p(\gamma - 1)}{k_B} \tag{2.15}$$

where the change in temperature dT is defined in terms absorbed energy U, Boltzmann constant k_B, proton mass M_p, mean molecular mass u and specific heat ratio γ (for Hydrogen).

2.5.3 Testing and Validation

The STARRAD module was tested and validated within the GASOLINE code (Wadsley, Stadel, and Quinn, 2004). GASOLINE is a multi-purpose parallel SPH code, based on TreeSPH (Hernquist and Katz, 1989), built as an extension to the Pkdgrav parallel N-body code. Already utilised for testing of other modules of the DIAPHANE library, GASOLINE is an appropriate choice for validation of STARRAD.

The validation test consists of a sun-like star enveloped in a gaseous disk. The disk is evolved statically, i.e. without hydrodynamical accelerations, from an initial disk shown in Figure 2.18 (top/bottom left). Initial conditions are generated using an iterative procedure that enforces initial hydro-static equilibrium by taking into account all forces (pressure, gravitational and centrifugal, see Rogers and Wadsley (2011)). The absorption coefficient in this context can be defined using a Rosseland mean opacity map as detailed in (D'Alessio, Calvet, and Hartmann, 2001).

As STARRAD computes only heating, some form of cooling is required in order to evolve to thermal equilibrium. To test in a standalone environment, STARRAD is paired with a standard cooling module within GASOLINE (Galvagni et al., 2012; Szulágyi, Mayer, and Quinn, 2017). This cooling model utilised a local optical depth approach which recovers the optically thick and optically thin limits of the cooling rate, essentially equivalent to the diffusion limit employed by FLD methods for large optical depths (e.g as used in the FLD module of Reed, Dykes, et al. (2018)). This takes advantage of the optical depth calculation performed by STARRAD.

Figure 2.18 (right) shows the specific irradiation of the disk as calculated by STARRAD, i.e. the energy absorbed per unit time per unit mass, along with a temperature map after heating; the dense inner disk is clearly shielding the outer disk in the midplane. The temperature profile in Figure 2.19 demonstrates that the STARRAD module combined with cooling quickly reaches a steady-state to reasonable values for protoplanetary disks. The incline in temperature in the 6-10 AU (Astronomical Unit) range results from high absorption reflecting the optical depth profile of the initial conditions. A plot of ray absorption in Figure 2.20 confirms negligible absorption within a ~6 AU radius, matching the density profile seen in Figure 2.21. Beyond the 9 AU radius, fast cooling results in the subsequent temperature decline, this is expected to be more moderate when coupled with FLD.

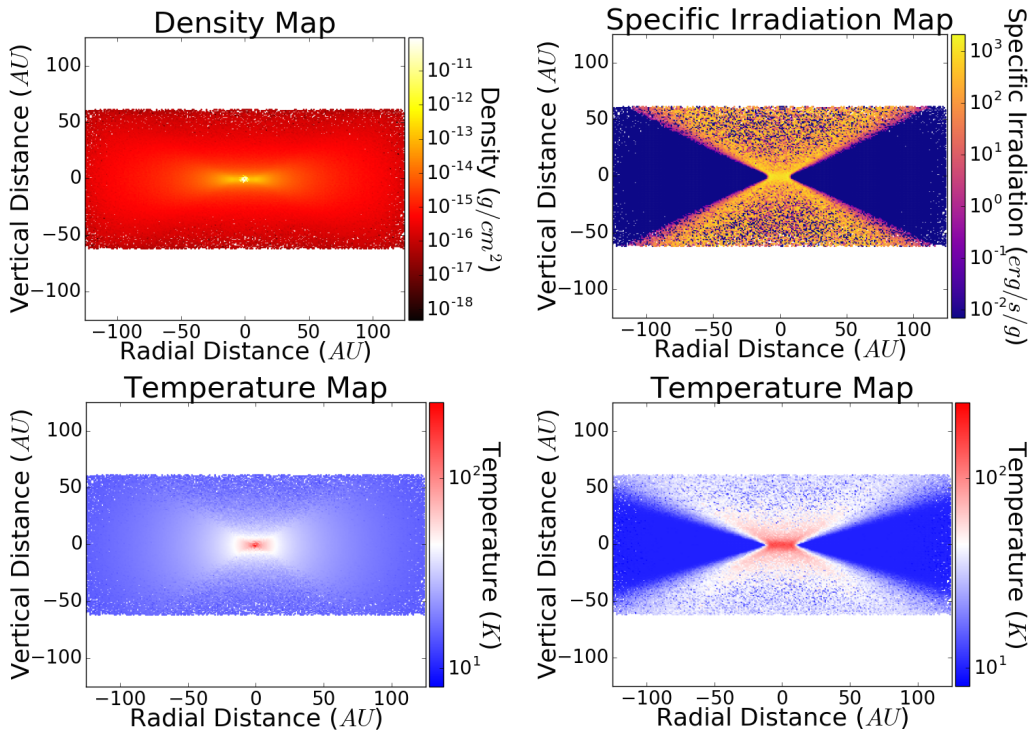


FIGURE 2.18: Colour coded maps representing a STARRAD static irradiated disk test; initial density map (left, top), initial temperature map (left, bottom). Specific irradiation (energy per unit time per unit mass) during the first step (right, top), and temperature after 300 years evolution (right, bottom).

2.5.4 Parallelisation

STARRAD has been initially parallelized through use of the Machine Dependent Layer (MDL) library, which serves as an abstraction layer for parallelism used by GASOLINE). A very brief overview of the parallelisation mechanism follows.

The parallel STARRAD model evenly distributes ray groups across processes using a ray indexing scheme given by HEALPix that maintains spatial coherency of rays across processes. Buffers for rays are allocated using an MDL allocation routine, such that the ray memory can be managed by the MDL library.

When a process writes optical depth contributions or reads the intensity field for a series of ray segments, they are fetched on demand from the remote process who owns the associated ray index. of remote read/writes. Stages 1 and 3 of the algorithm require a lot of communication as ray segments and written with optical depth contributions, and read for intensities respectively. STARRAD makes use of fast caching mechanisms available in MDL to mitigate the communication overhead. Stage 2 is an requires no communication, as rays are integrated locally on each each process.

The current validation and testing scheme confirms correct parallel execution, but further work will be necessary to understand performance and scaling of this approach.

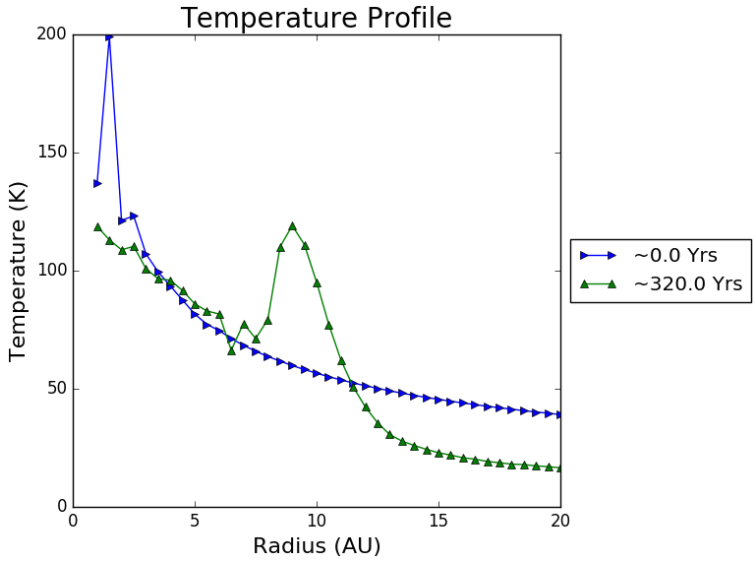


FIGURE 2.19: Temperature profile of the STARRAD static irradiated disk over ~300 years of evolution, corresponding to ~30 disk orbits at 10AU.

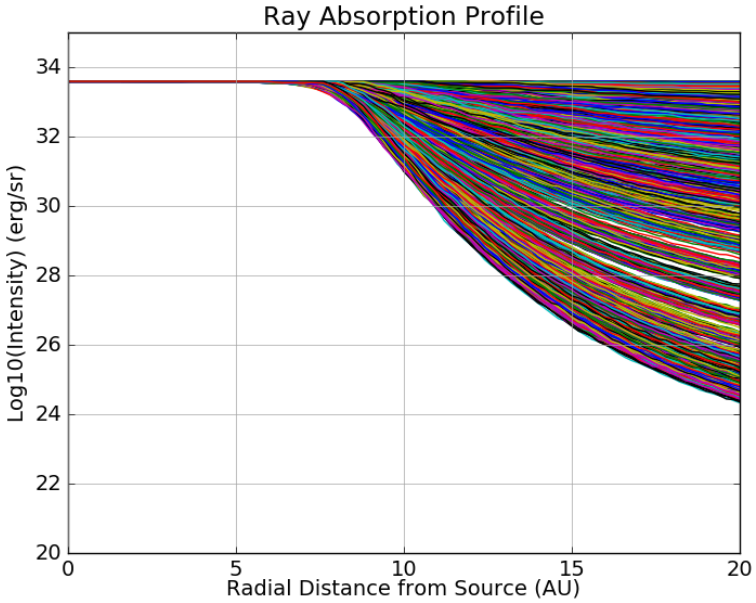


FIGURE 2.20: Absorption profiles for rays. The initial low density below 6AU provides negligible absorption, while above 6AU the energy (per unit solid angle) of each ray steadily declines as it is absorbed by the denser regions of the disk.

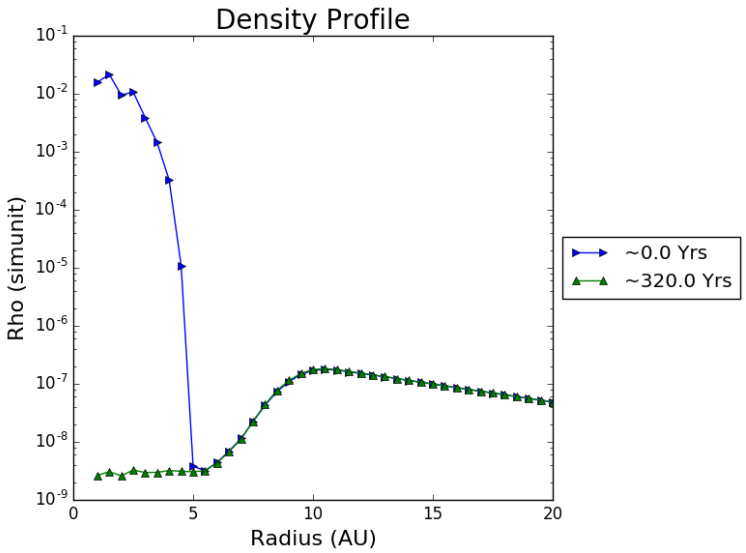


FIGURE 2.21: Density profile of the STARRAD initial conditions and irradiated disk after ~300 years of evolution. This does not change significantly after an initial readjustment phase in the sparse inner region of the initial disk.

2.5.5 Discussion

This section is divided in two. Firstly, discussion from an astrophysical perspective is presented (Section 2.5.5.1), and secondly comments from a graphical perspective are given (Section 2.5.5.2).

2.5.5.1 In the Context of STARRAD

In terms of next steps, the accuracy of STARRAD has been verified in terms of energy conservation and heating rates, a more rigorous verification is necessary within a robust astrophysical environment. While the current tests utilize a static disk, to simplify validation of the radiative transport, STARRAD is also able to run with hydrodynamics; indeed in the current run the hydrodynamics are active (recomputing density/pressure at each step) however accelerations are turned off, hence a static disk. Note that the initial adjustment of the disk from time t=0 is due to initialization procedure (described above) applied to enforce hydrostatic equilibrium to the disk, whose inner region consists of few particles due to a hole by construction (Fig 2.21, $<$ 5AU).

Whilst there are existing particle-based methods for treatment of radiative transport in astrophysics (e.g. (Ritzerveld, Icke, and Rijkhorst, 2003), (Pawlik and Schaye, 2008)), STARRAD takes a unique and simple approach to treat direct lighting, inspired by volume splatting methods in computer graphics (discussed in Section 2.5.5.2). This simple approach is viable as STARRAD is part of a wider group of radiative transfer algorithms in the DIAPHANE library, and intended to be used in a hybrid manner. For example, the first physical application of STARRAD will be in the planet

forming disk example introduced in Section 2.5.1. In order to self-consistently model the evolution of gas surrounding a protostar, STARRAD will be coupled to the Flux Limited Diffusion (FLD) module of the DIAPHANE library in a manner similar to Klassen et al. (2014), where each time-step exploits multiple methods of calculating radiative transfer. Flux-limited diffusion is well-suited for the optically thick environments of forming stellar systems, and enables the radiation field to be split into direct and indirect components. In this case, direct heating of the gas by the protostar's light is modelled with STARRAD and FLD is used to follow diffusion of that heat in the optically thick disk, allowing higher accuracy in the evolution of the gas properties (Murray et al., 1994).

The exploitation of STARRAD, and this coupled model, in a robust astrophysical environment is the next step for the STARRAD work, however is beyond the scope of this thesis.

2.5.5.2 In the Context of Volume Rendering

This section has presented a simple astrophysical approach to radiative transfer, highlighting two clear overlaps with volume rendering that could be beneficial for the physical motivation of Splotch.

The STARRAD algorithm essentially computes a light volume via splatting, in a similar manner to Nulkar and Mueller (2001), with which to determine volume heating. The key difference is that the sheet buffer in this case is made up of concentric pixel spheres (each sphere i consists of all ray segments at location i along the rays). With this in mind, a potential further application of STARRAD could be to couple with Splotch, computing direct lighting for cases where one, or a small number, of light sources illuminate the scene. This would be an effective means of adding shadows to Splotch for visualising scenarios such as the gaseous disk example used for STARRAD testing and validation.

Furthermore, the STARRAD implementation in GASOLINE makes use of a precomputed opacity map, that relates density and temperature to an opacity coefficient (K). This map could simply be exploited in the Splotch transfer function, replacing the colour table lookup, as a physically motivated absorption definition for gas particles.

2.6 Summary

This chapter addresses the objective **O.1**: *'Understand and exploit domain specific data and knowledge to physically motivate volume rendering for astronomical data'*.

Following an overview of visual realism and physically motivated visualisation for astronomy, along with the existing optical model of high performance visualisation software Splotch, this work presents an improved model to address frequency dependent absorption and emission. A novel galaxy modelling and visualisation approach is then presented in Section 2.4, exploiting the new optical model to build and

visualise 3D galaxy models including dust lanes using astronomical galaxy observations and theoretical data. The pipeline is exhibited through the well known spiral galaxy M83, showing high quality and physically realistic results and demonstrating the efficacy of this new modelling and visualisation approach. Section 2.5 moves beyond the visualisation to demonstrate the strong overlap between astrophysical simulations and computer graphics in terms of radiative transport. A particle based radiative transfer algorithm STARRAD is implemented in the context of a general purpose radiative transport library, and validated through a protoplanetary disk simulation exploiting astrophysical SPH code GASOLINE, allowing the proposal of knowledge transfer back to computer graphics relating to coupled lighting and precomputed opacity maps. Part of the work present in this chapter, specifically Section 2.5, is presented within the context of astrophysical simulations in Reed, Dykes, et al. (2018).

In view of the question posed: **Q.1**: *'How can domain expertise in astronomy be used to physically motivate volume rendering of astronomical data?'*, this chapter has demonstrated the use of astronomical domain expertise to physically motivate both the modelling and rendering of astronomical data from both theoretical and observational sources through the above research contributions. However, as described in Chapter 1, astronomical data is typically very large in volume, and requires high performance, parallel, and remote approaches to visualisation. The following chapters move beyond visual quality to address the growing hardware complexity, increasing data sizes, and modern research environments for scientific visualisation in astronomy.

Chapter 3

Heterogeneous High Performance Visualisation

This chapter aims to answer the question **Q.2**: *'How can high performance visualisation cope with modern heterogeneous high performance computing systems?'*, by addressing the objective **O.2**: *'Analyse and exploit emerging HPC architectures in the context of high performance visualisation'*. To this end, Sections 3.1 and 3.2 provide a brief overview of the increasing popularity of multi-core and many-core accelerator hardware in HPC systems and the challenges presented; the following sections (3.3, 3.4, and 3.6) aim to address these within the context of high performance visualisation via a series of algorithmic implementation and optimisation approaches for scientific visualisation algorithm Splotch in a variety of heterogeneous and distributed memory HPC environments. Section 3.7 extracts the general and specific methods used in the previous sections to define methodologies for the implementation or porting of general algorithms, and volume splatting specific algorithms, on future architectures. The chapter concludes with a brief summary of research contributions (Section 3.9).

3.1 The Emerging HPC System

The modern day supercomputer is a result of decades of research and development into increasingly parallel computing. Early supercomputers of the 1950s, 60s, and 70s (e.g. the CDC machines and the Cray 1) typically executed a single instruction stream sequentially with vector processors; however in the 1980s this paradigm shifted to multi-processor designs, such as the Cray X-MP starting with two CPUs and ranging up to eight by the end of the decade (Hoffman and Traub, 1989). The trend of increasing numbers of parallel CPUs continued throughout the 90s, culminating at the end of the century with the 9632 processor ASCI Red Intel system that topped the November 1999 Top500 list (Meuer et al., 1999), an on-going compilation of the worlds most powerful supercomputers[1]. During this period, and primarily due to economic forces (Rajovic et al., 2013), the custom vendor-built vector processors in HPC machines were replaced by commodity such as SPARC and MIPS that

[1] https://www.top500.org/

were based on the Reduced-Instruction-Set-Computer (RISC) instruction set architecture (ISA).

Typically, these CPUs had only one processing core, and HPC systems achieved high performance by increasing the number of CPUs working in parallel. The performance of each individual CPU also grew, following Moore's Law (Moore, 1965), and higher single-core performance was achieved through increased transistor density, clock frequencies, and pipeline complexity. During the early 2000s, RISC architectures in HPC were mostly replaced by Intel's x86 architecture (Rajovic et al., 2013), which at the time was still based on a single-core. However, single-core performance is constrained by a series of limitations in areas such as energy consumption, dissipation, and power density (Markov, 2014). As such, in an effort to extend Moore's Law and continue to scale CPU performance, a trend of *multi-core* CPUs emerged (Geer, 2005). This trend allowed CPU performance to continue to scale; however, physical limitations such as minimal transistor size, or practicality limitations such as acceptable power to performance ratios, cost, and cooling requirements, are likely to cause the end of Moore's Law in its current form and change the way CPU chips are developed (Colwell, 2013) (Courtland, 2015). While its longevity remains unclear to this day (Eeckhout, 2017), the trend in increasingly parallel processor design is clear, leading to even more cores per chip exploiting widening vector registers for instruction level parallelism. This trend has led not only to tens of cores in multi-core CPUs, but also the introduction of *many-core* architectures exploiting tens, hundreds, or even thousands of cores.

In the latter half of the 2000s, and concurrently with the on-going parallelisation of the general purpose CPU, a trend of heterogeneity emerged in the form of HPC systems incorporating additional many-core processors known as *accelerators* and/or *coprocessors*. This concept of highly parallel accelerator hardware was not new at the time, for example the Distributed Array Processor (Manning, 1989), however until the mid 2000s this type of hardware was not generally included in high performance systems. The first entry to appear in the Top500 list that exploited a commercial accelerator card was the TSUBAME Grid Cluster in June 2006 (Meuer et al., 2006). This NEC/Sun machine featured 360 ClearSpeed CSX600 accelerator cards and was hosted at the Tokyo Institute of Technology. Later, in June 2008, the list was topped by RoadRunner, the first HPC machine to achieve a sustained 1.0 petaflops on the Top500 LINPACK[2] benchmark. Roadrunner featured a hybrid design consisting of AMD Opteron 2210 processors combined with IBM PowerXCell 8i processors based on the Cell co-processor architecture. This trend of *heterogeneous computing* continued with HPC machines incorporating commodity GPUs as accelerators, the rising importance of which was noted by Buchty et al. (2011) in their survey of heterogeneous computing which provides a picture of heterogeneity in HPC at the end of the first decade of the new millennium. More recently, this has progressed to the on-going development of a series of custom designed many-core

[2] http://www.netlib.org/benchmark/hpl/

chips (e.g. Intel Xeon Phi, Sunway SW26010 and increasingly powerful data centre GPUs). As of June 2018, nine of the top ten supercomputers in the world are either heterogeneous (exploiting accelerators/coprocessors) or entirely hosted on many-core processors to achieve top computational performance.

Whilst the future of the HPC industry is not clearly defined, the pinnacle of modern HPC performance is being achieved through many-core and heterogeneous computing, and looking at the trends over the past decades it is likely some form of heterogeneous computing will continue to be prevalent in emerging HPC systems (Kogge et al., 2008; Giles and Reguly, 2014). Therefore, the ability to fully exploit new heterogeneous and many-core architectures is of paramount importance towards achieving optimal performance on modern HPC systems. Exploiting these emerging hybrid architectures is non-trivial however, due to the challenges presented by mixed hardware computing and the increasing levels of architectural parallelism.

3.2 Exploiting Massively Parallel Systems

One of the key challenges presented by the trend outlined in the previous section, is the need for new algorithms and approaches to effectively exploit hardware in a heterogeneous HPC system (Kogge et al., 2008; Kowalkowski, 2014; Reed and Dongarra, 2015; Geist and Reed, 2017).

There is a variety of hardware in a HPC system that may be considered heterogeneous or an accelerator. Cascaval et al. (2010) introduce a taxonomy of accelerator hardware based on a variety of characteristics including architecture, programming models, addressing modes, parallel granularity and synchronisation types; including a discussion on the available programming models for such systems. This taxonomy extends from tightly-coupled accelerators such as typically on-chip functional units (floating point and vector units), to a variety of more loosely-coupled accelerators. For the remainder of this thesis, the term accelerator refers to programmable architecture hardware devices coupled to traditional CPU hardware, typically via the I/O bus (e.g. PCIe), and more specifically that commonly found in current HPC systems: General Purpose Graphics Processing Units (GPGPUs), and the Intel Xeon Phi.

The challenge of effectively exploiting accelerator hardware can be decomposed into two parts; algorithm and implementation. New algorithms must be designed, or old ones adapted, to exploit the growing parallelism in accelerator hardware. This task is made difficult by the on-going *concurrency challenge*: 'to benefit from the continued increase of computing power according to Moore's law, application software must support concurrent execution by multiple processor cores' (Hwu, Keutzer, and Mattson, 2008). As a result of the paradigm shift from single to multiple-cores, also known as *the concurrency revolution* (Sutter and Larus, 2005), most high performance codes already exploit parallelism at multiple granularities to achieve performance on HPC systems (for detail on granularity of parallel programming see Hwang

(1992)). However, many-core architectures underpinning current accelerators for HPC typically necessitate a more explicit approach to multi-grain parallelism (see characteristics below).

At the implementation level, effective exploitation of heterogeneous systems means achieving high computational performance on specific hardware, and performance-portability across the variety of available accelerator hardware (e.g. (Cascaval et al., 2010; Liu et al., 2012)) and associated programming models (Diaz, Muñoz-Caro, and Niño, 2012), along with future as-yet-unknown hardware. The work presented in the following sections improves performance for specific hardware by focusing on architectural characteristics that are expected to prevail in future iterations of heterogeneous systems as outlined in the previous section. To summarise:

Increased core count: It is likely that future HPC architectures, heterogeneous or not, will retain a high core-count. A higher number of low-power cores necessitate increased *task parallelism*, however this can be difficult to accommodate due to problems relating to parallel overhead, complex data dependencies, program structure, or limitations imposed by Amdahl's Law (Amdahl, 1967).

Wider Vector Registers: The widening of vector registers require increased *data parallelism* for Single Instruction Multiple Data (SIMD) type operations. This can be crucial to accelerators such as the GPU and Xeon Phi that rely on high data parallelism to achieve the computational throughput required for peak performance (as demonstrated in the following subsections).

Complex memory hierarchies: Accelerators typically have their own cache hierarchies and on-board memory, which may be complimented by different types of memory such as GDDR5 on a GPU or MCDRAM for Intel Xeon Phi Knights Landing (KNL) models. Algorithms must be capable of addressing increased memory complexity with both deeper and wider memory hierarchies and increased NUMA effects between processor cores (Tate et al., 2014).

High performance visualisation algorithm Splotch is well suited as a candidate to explore these characteristics and compare the performance achievable on modern many-core and accelerator architectures, as outlined in 1.4. The following sections detail specific case studies exploiting accelerator hardware in HPC systems. In Part I (Section 3.3), the presented work details the process of porting and optimising for the many-core Intel Xeon-Phi Knights Corner (KNC). In Part II (Section 3.4), the presented work addresses the rapid evolution of existing accelerators, building on the previous work of (Rivi et al., 2014) to exploit new GPU architectural developments. Finally, Part III (Section 3.5) compares the performance of each of the implementations.

3.3 Accelerators Part I: Xeon Phi

The Intel Xeon Phi coprocessor (Chrysos, 2012) is based on the Many Integrated Core (MIC) architecture (Intel, 2013b), which is envisaged to provide, on suitable classes of algorithms, outstanding performance with power consumption being comparable to standard CPUs.

Many developers have been exploring the possibility of using the MIC based products to accelerate their software, as can be seen in the Intel Xeon Phi Applications and Solutions catalogue (Belinda, 2014); high performance visualisation software developers are also on board with efforts being made to extend VisIt[3], an open source scientific visualisation tool, to exploit the MIC architecture at the Intel Parallel Computing Center at the Joint Institute for Computational Sciences between the University of Tennessee and Oak Ridge National Laboratory. Developers are enticed not only by the large potential for compute power, but also a key advertised feature of the Xeon Phi, and MIC architecture: the ability for developers to work with regular programming tools and methods they would use for a standard Xeon (or indeed, other processors). This includes use of parallel APIs and runtimes such as MPI and OpenMP from standard C++ and Fortran source code, while also offering the possibility of using more specialized options for parallelism such as Intel's Cilk Plus[4].

Current experiences implementing algorithms or porting pre-existing parallel codes to the Xeon Phi show many successes optimising specific kernels and improving performance as compared to unoptimised kernels on Xeon Phi (e.g. (Borovska and Ivanova, 2014; Gaburov and Cavecchi, 2014; Deslippe et al., 2015)). Despite this, when comparing overall program performance against that achievable on a node with standard Xeon CPUs, authors may achieve similar or lower performance to the CPU (e.g. (Elena and Rungger, 2014), (Reid and Bethune, 2014)), and it is not surprising considering the recent introduction of the architecture and the reported difficulties in achieving performance Fang et al., 2014. This, however, is in the process of changing as more efforts are made to utilise the architecture with improvements showing in all areas as the technology matures. One particular success story of note is the exploitation of the Tianhe-2 supercomputer (at the time exploiting Xeon Phi) to achieve peta-scale performance with an earthquake modeling simulation (Heinecke et al., 2014).

3.3.1 Overview of the Xeon Phi

The idea behind MIC is obtaining a massive level of parallelism for increasing throughput performance in power restricted cluster environments. To this end Intel's flagship MIC product line, the Xeon Phi, contains roughly 60 cores on a single chip, dependent on the model, and acts as an accelerator for a standard Intel Xeon

[3] https://visit.llnl.gov
[4] http://www.cilkplus.org/

processor. Programs can be executed natively by logging via ssh into the device, which hosts a Linux micro-OS, or by using the device through one or more MPI processes in tandem with those running on the Xeon host (symmetric mode). Alternatively users can offload data and portions of code to the coprocessor via Intel's Language Extensions for Offload (LEO), a series of pragma based extensions available in C++ or Fortran. For a detailed technical description of the processor's architecture, the reader is referred to the Xeon Phi white paper (Chrysos, 2012). Here, a very short overview of the main features of the *Knights Corner* model is provided.

Each core has access to a 512 KB private fully coherent L2 cache and memory controllers and the PCIe client logic can access up to 8 GB of GDDR5 memory. A bi-directional ring interconnect brings these components together. The cores are in-order and up to 4 hardware threads are supported to mitigate the latencies inherent with in-order execution. The Vector Processor Unit (VPU) is worth mentioning due to the innovative 512 bit wide Single Instruction Multiple Data (SIMD) capability, allowing 16 single precision (S.P.) or 8 double precision (D.P.) floating point operations per cycle, with support for fused-multiply-add operations increasing this to 32 S.P. or 16 D.P. floating point operations.

3.3.2 Splotch on the MIC

3.3.2.1 Implementation

The Splotch algorithm (as introduced in Section 2.3.1) ported to the Xeon Phi uses the offload model, illustrated in Figure 3.1. Whilst the executable runs on the Xeon host, data and processing are offloaded to the device via Intel's LEO. The ability to run already OpenMP and MPI based programs on the Xeon Phi means the swiftest approach to enable Splotch to run effectively on the device is to modify the current implementation, as opposed to moving to another software paradigm such as Intel's Cilk Plus which may provide additional features but would involve a more thorough re-write of the algorithm, future work is envisioned to also explore these paradigms.

The rasterisation phase consists of a highly parallel 3D transform, projection, clipping, and colour assignment on a per-particle basis. These are split into two kernels, the transformation (including projection and clipping) and the colouring. Transform parameters are precomputed asynchronously, and work is distributed amongst threads via OpenMP parallel for-loops. The already highly parallel nature of these loops meant that no significant algorithmic modifications were needed in order for the code to run, however both are optimised through use of manual and automatic vectorization to provide a performance boost for this phase (see Section 3.3.2.2).

The rendering phase consists of a pre-render stage, and a ray-casting render stage. single kernel in which image pixels are subdivided in a two dimensional grid of tiles that are distributed amongst threads. The size of these square tiles are defined by a runtime parameter (see Section 3.3.3). During the pre-render stage, a

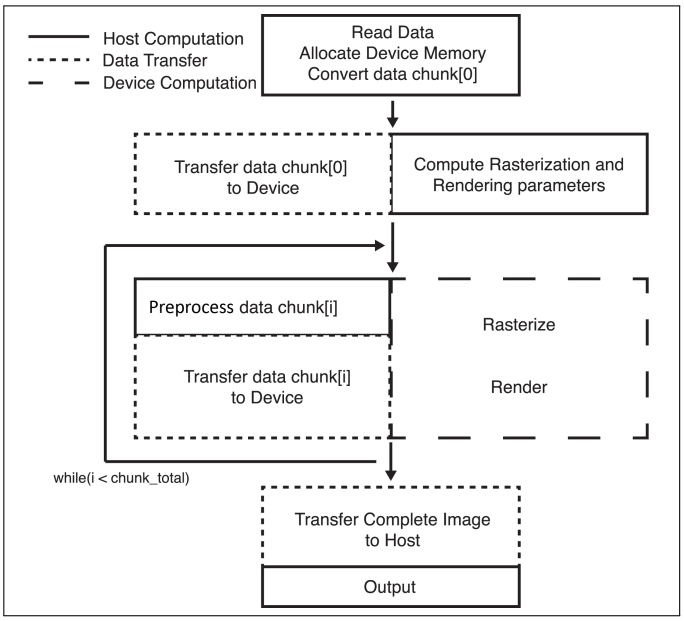


FIGURE 3.1: Model illustrating execution flow of the offloading implementation.

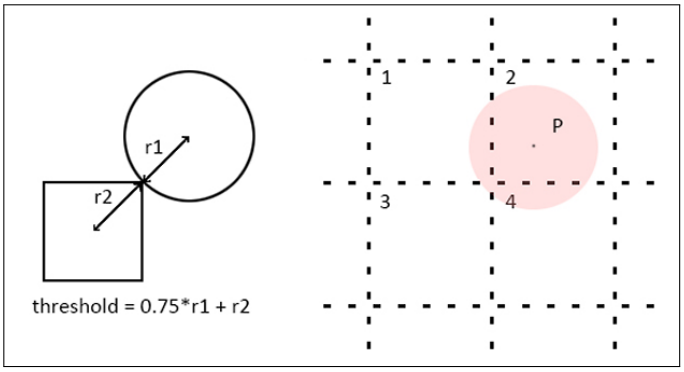


FIGURE 3.2: *Left:* Distance threshold definition below which a particle is considered to affect a tile. *Right:* Example of a particle projected onto a 2D tiled image. In this case, the particle is considered to affect tiles 1, 2, and 4, as the overlap between particle P and tile 3 is negligible.

list of particle indices is created for each tile, indicating all particles affecting that tile (see Figures 3.2 and 3.3). Following this, each thread renders a full list by solving the radiative transfer equation along line-of-sight rays, and retrieves another in round-robin fashion; in this way pixel access is kept local to each tile and not shared between threads, avoiding concurrent access and race conditions. Finally when all chunks of data have been processed and accumulated, the resultant device image is copied back to the host for output. The original OpenMP rendering process described in (Rivi et al., 2014) has not been conceptually modified, rather the implementation has been optimised for Many Integrated Core.

Asynchronous data transfer is supported on the device via *signal* and *wait* clauses, provided as part of the Intel C++ LEO language extensions, to trigger and

```
begin parallel block
  get_particles_for_thread(threadID, start, end)
  for each p in particles[start] : particles[end]
    get_affected_tiles(range, tile_size, p.x, p.y, p.radius)
    threshold = 0.75*p.radius + sqrt(2)*tile_size/2
    for each x in tiles[range.min_x] : tiles[range.max_x]
      cx = get_center(x)
      for each y in tiles[range.min_y] : tiles[range.max_y]
        cy = get_center(y)
        distance = length((p.x, p.y), (cx, cy))
        if(distance < threshold)
          indices_vector[thread_id][x][y].push_back(p.index)
        end if
      end for
    end for
  end for
end parallel block
```

FIGURE 3.3: Pseudocode illustrating the construction of vectors of particle indices which affect the tiles of a decomposed image.

monitor asynchronous transfers in a parallel environment. To facilitate the visualisation of datasets potentially much larger than the memory capacity available, while minimizing overhead due to data transfers, a double buffered scheme to overlap computation and data transfer has been implemented using these clauses. The scheme creates two storage buffers; data from the host is copied to the first buffer, and while this data is processed the second buffer is asynchronously populated. The second buffer is then processed while the first buffer is asynchronously replaced with a new set of data. This loop can continue indefinitely while data is available in host memory, thereby solving the problem of limited device memory without costly delays in processing due to waiting for a full device buffer to be repopulated. The efficacy of this approach is dependent on the ratio of computation to Input/Output (I/O), i.e. more time must be spent rendering particles on the device than transferring data. The minimum computational cost a particle can incur in the scene is in the case where it affects a single pixel, a point-like particle. Referring to the results in Section 3.3.4.2, a rough estimate for the minimum computational time per-particle (smallest average radius) is 2.2E-8 seconds. The tests show on average 9.2E-9 seconds per-particle for data transfer. As such, the approach is beneficial even in the least computationally expensive type of scene.

3.3.2.2 Optimisation

The three main kernels of the code, transformation, colouring, and rendering, consume the majority of processing time. These are optimised by targeting a set of key problems known to cause performance issues with the Xeon Phi, as laid out in various resources for programming such device[5], and generally highlighted in Sections

[5]For example: https://software.intel.com/en-us/mic-developer

3.1 and 3.2. Memory usage, data transfer, vectorization, and general tuning are discussed in the following subsections.

Two key tools were employed to analyze performance and identify target areas for optimisation. The first is Intel VTune Amplifier XE, a profiling tool capable of measuring, analyzing, and visualising performance of processing, both offloaded and native, on the Xeon Phi. Amongst other features, it simplifies the process of evaluating the benefit of manually inserted intrinsics in comparison to those generated by the compiler.

The second tool adopted for tuning is the Performance API (PAPI)(Mucci et al., 1999). This API assists direct measurement of hardware events on the device. In particular the ability to target a select set of statements with in-code hooks can be used for fine grained capture of hardware events in a large codebase. Moreover PAPI provides a high level of hardware event control in a lightweight form, without the auto-analysis and visualisation options of a more fully featured tool such as VTune Amplifier. A small wrapper was created to facilitate use of PAPI in Intel's offload mode, which is available on Github with a sample benchmark and setup instructions[6].

Optimisation I: Memory Usage

Addressing both the memory hierarchy, and increased core counts, as highlighted in Section 3.2, this phase of optimisation targets dynamic and threaded memory allocations.

Cost of dynamic memory allocation on the Phi is relatively high (Intel, 2013a), so in order to minimize unnecessary allocations buffers are created at the beginning of the program cycle and reused throughout. It was not possible to asynchronously allocate memory using offload clauses (allocating directly with a LEO clause as opposed to offloading a call to *malloc*), and so overheads incurred allocating these buffers cannot be mitigated by overlapping allocation with host activity. Use of the MIC_USE_2MB_BUFFERS environment variable forces buffers over a particular size to be allocated with 2MB pages rather than the default 4KB, which improves data allocation and transfer rates and can benefit performance by potentially reducing page faults and translation look-aside buffer (TLB) misses (Intel, 2012b). These experiences showed a single process offloading to the device, and reserving large buffers, can allocate memory roughly 2-2.5x faster having set this environment variable appropriately. This reduces the cost for initial memory allocation (column two vs. three in Fig. 3.6) and decreases L1 cache misses, however detrimentally affects kernel execution speed by increasing the L1 TLB miss ratio (Table 3.1). In this case, the environment variable is appropriate for single-image visualisation, where initial memory allocation is a large proportion of the overall time. For movie-generation, where initial allocation occurs only once, the benefit of faster allocation is outweighed by the

[6] https://github.com/TimDykes/mic_papi_wrapper

Derived Metric	4KB Pages	2MB Pages	% Diff.	Effect
Per-Thread CPI	4.83	4.95	2.48	negative
L1 Misses	54.0E+06	42.0E+6	-20.3	positive
Estimated Latency Impact	10.4E+03	14.5E+03	39.4	negative
L1 TLB Miss Ratio	0.001	0.014	1300	negative
L1 TLB Misses per L2 TLB Miss	20.0	105	425	negative
Transform kernel time/s	0.263	0.367	39.5	negative

TABLE 3.1: The effect of 2MB vs. 4KB page allocation on overall cache and TLB usage, with subsequent effect on execution speed for the offloaded transformation kernel, obtained via Intel VTune. Both positive and negative values of the percentage differences can indicate a performance improvement, depending on the parameter. Beneficial factors are indicated as "positive" in the last column.

increased kernel execution time and it is advantageous to allocate with 4KB buffers as the default.

A notable issue for the many-core architecture is scalable memory allocation. When working with dynamic memory, for example through use of Standard Template Library (STL) containers, parallel calls to *malloc* can be serialized (Intel, 2012d). In this case, memory allocation issues are seen in the pre-render phase, where the target image is geometrically decomposed into n tiles and for each tile a vector of particle indices is generated (as described in Fig. 3.3). This requires each OpenMP thread to create n arrays, which are then filled with particle indices to be rendered. Each vector requests more memory as it reaches capacity, resulting in significant stalls caused by simultaneous allocations from different threads.

For the Xeon host no attention is paid to potential memory allocation stalls, due to the relatively low number of threads, typically 16 or less. The number of threads active on the many-core architecture however is often much higher, resulting in thousands of dynamic arrays across many threads, necessitating a scalable solution to memory allocation. It is already possible to achieve sufficiently scalable memory allocation in parallel environments through use of external libraries, e.g. *ptmalloc*(Gloger, 2006) or Intel Threading Building Blocks *scalable_allocator*[7]. However, in order to solve this problem without introducing dependencies on external libraries, a custom memory management solution is implemented consisting of a template array (*array_t*) and a memory allocator. Each thread is provided with an instance of the allocator, which asynchronously allocates a user-defined subset of storage from the device memory in the initial setup of the program. This "pooled" memory is then sub-allocated on request to local arrays, which greatly reduces risk of clashing calls to *malloc* from differing threads. The allocator implements bi-directional coalescence in order to minimise fragmentation, maintains the requested alignment for allocations, and if unable to provide the requested amount of memory it will request this via a standard aligned allocation.

Performance is compared for an STL vector container with and without Intel's

[7] https://www.threadingbuildingblocks.org/tutorial-intel-tbb-scalable-memory-allocator

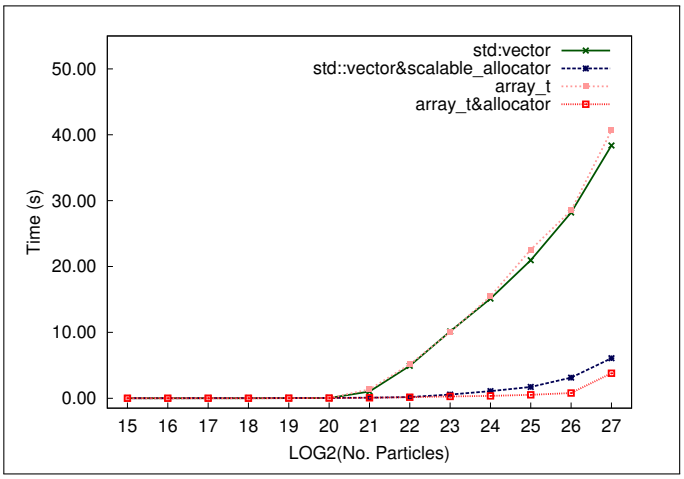


FIGURE 3.4: Comparing performance of an *std::vector* with and without Intel's scalable_allocator against a custom array implementation (*array_t*) with and without pooled memory allocation.

scalable_allocator, against the custom vector implementation *array_t* with and without the pooled memory allocator. It can be seen in Fig. 3.4 that performance for the STL container, and custom array implementation using standard aligned allocation methods (i.e. *_mm_malloc()*), decreases significantly as the particle count rises above 2^{20}, or roughly 1 million. The *array&allocator* retains high performance as all containers local to a thread are allocated memory from a dedicated per-thread memory pool. In the largest cases, where allocations begin to exceed local memory pools and are only limited by device memory, the custom allocator performs slightly better; this is in part due to faster 2MB page allocation, which does not appear to benefit Intel's scalable allocator. The effect of using this allocator in Splotch can be seen in Fig. 3.6, reducing the dominating pre-render stage in column one to be negligible in column two.

Overheads in dynamic allocation and data transfer can incur a penalty when running a single host process offloading to the device. In order to minimize these penalties, the MPI implementation of Splotch is employed. Multiple MPI processes on the host are each allocated a subset of the device threads to exploit. In this way, the device is subdivided amongst the host MPI processes allowing to minimize overheads in data transfer, allocation and processing providing a noticeable performance increase, further details of which are given in Section 3.3.4 and similar experiences can be seen in (Borovska and Ivanova, 2014).

Optimisation II: Vectorization

Addressing the widening vector registers, as highlighted in Section 3.2, this phase of optimisation targets both manual and automatic vectorization.

The large 512 bit wide SIMD capability of the MIC architecture is exploited through vectorization carried out both automatically by the compiler, and manually using Intel intrinsics which map directly to Intel Initial Many-Core Instructions

(IMCI) (Intel, 2012c). The vectorization process and performance gains are discussed for the three main kernels.

The roto-translation and filtering stage of rasterisation is a fairly simple and highly parallelisable geometric transformation, there are no data interdependencies and minimal branching. It was modified to enable auto-vectorization through a series of steps described in Intel's Vectorization guide (Intel, 2012a), and feedback from the compiler vectorization report. While the auto-vectorization is not often instantly applicable, in simple cases such as this the modifications are fairly trivial compiler pragmas and correct alignments.

Conversely, the colouring stage of rasterisation involves a per-particle inner loop through an external colour look-up table. This causes the compiler not to auto-vectorize the most suitable loop that in this case is the outer one through particles. As a solution, the outer loop is iterated in increments of 16 (i.e. the device S.P. SIMD vector capability) and manually vectorized; the remainder loop (if any) is processed in a non-vectorized fashion. Performance of this modification was analysed with PAPI; in Table 3.2, a set of native hardware events plus derived metrics have been measured before and after the insertion of manual intrinsics. It can be seen from this example that the manual intrinsics doubled the speed of the kernel, while the measured FLOP/s increased by 60%.

Looking more closely at Table 3.2, contributors to the performance gain include the 20% decrease in instructions retired per thread (event: *instructions_executed*) coupled with a 30% drop in CPU cycles overall (event: *cpu_clk_unhalted*). The 7% rise in vector intensity (derived metric: *vpu_elements_active / vpu_instructions_executed*) indicates more of the vector elements in a VPU register were active on average during vectorized execution, leading to less instructions necessary. It is likely that the reformat of instructions involved in manual vectorization had a higher impact on the reduced cycle count (and therefore reduced time to solution) than the mildly higher level of vectorization indicated by the vector intensity metric. It should be noted that it can be difficult to interpret the meaning of hardware events when not accompanied by an increase or decrease in time, and so for more information on the capture, utilisation and derivation of hardware metrics the reader is referred to the relevant Intel optimisation guide (Intel, 2012e).

The rendering phase of the algorithm is difficult for the compiler to auto-vectorize due to a loop through non-consecutive particles followed by partially consecutive pixel updates. An image pixel's current RGB values is additively combined with the contribution from the current particle. This is calculated by multiplying the particle colour by a scalar contribution value representing the Gaussian spread over the distance between the pixel and the particle centre on each axis. For each pixel on the horizontal axis, a scalar contribution value is calculated per-pixel on the vertical axis, (illustrated in Figure 3.5). Only the inner vertical axis loop update contains consecutive memory accesses for vectorization. In this case the kernel was

Core Event	**Before**	**After**	**% Diff.**	**Effect**
`time`	0.16	0.07	-53.88	positive
`vpu_instructions_executed`	2.18E+09	1.49E+09	-31.53	positive
`vpu_elements_active`	9.00E+09	6.60E+09	-26.69	positive[a]
`cpu_clk_unhalted`	3.54E+10	2.31E+10	-34.72	positive
`instructions_executed`	8.08E+09	6.24E+09	-22.82	positive
`long_data_page_walk`	8.94E+03	7.99E+03	-10.61	positive
`L2_data_read_miss_mem_fill`	6.69E+06	1.11E+07	66.51	negative
`L2_data_write_miss_mem_fill`	1.15E+07	1.50E+05	-98.69	positive
Derived Metric				
Vector intensity	4.13	4.42	7.07	positive
GFLOP/s	55.82	88.73	58.97	positive
Per-Thread CPI	4.38	3.7	-15.42	positive
Per-Core CPI	1.09	0.93	-15.42	positive
Read Bandwidth (bytes/clock)	0.13	0.13	-4.73	negative
Write Bandwidth (bytes/clock)	0.07	0.09	28.89	positive
Bandwidth (GB/s)	12.57	13.46	7.07	positive

[a]Positive when the percentage reduction in `vpu_instructions_executed` is larger.

TABLE 3.2: Comparing various hardware events and metrics measured with PAPI before and after manual vectorization of the colourise kernel.

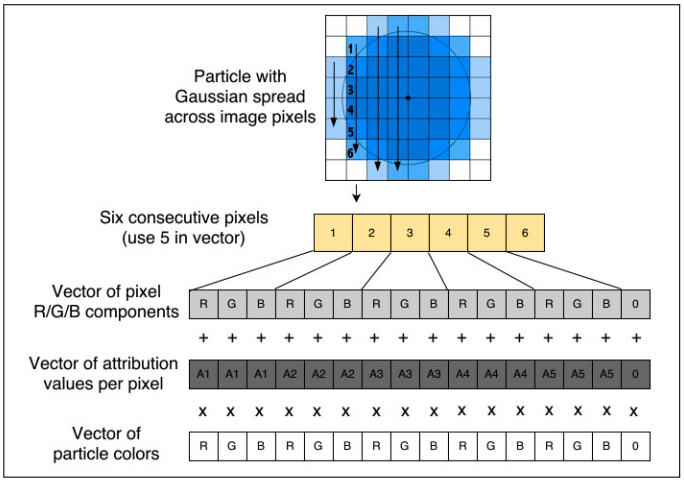


FIGURE 3.5: Vectorized update of up to 5 consecutive image pixels via Fused-Multiply-Add instruction.

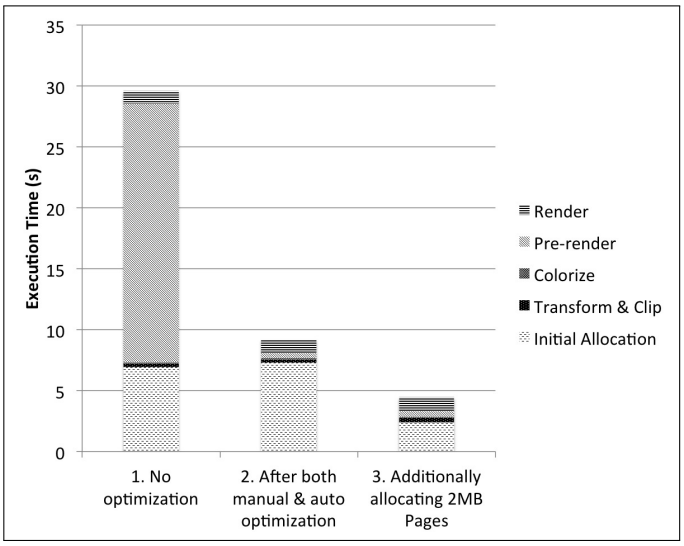


FIGURE 3.6: The effect of optimisation: Column 1 shows the code with no optimisation, Column 2 includes all optimisations discussed in Section 3.3.2.2, while column 3 includes 2MB page allocation which greatly reduces initial allocation cost while subtly increasing kernel execution times due to cache usage.

manually optimised through extensive use of intrinsics: five single precision particle RGB values (totaling 480 bits) and five scalar contribution values are packed into two respective 512 bit vector registers, V1 & V2; a third register, V3 contains 5 affected pixels, which are written simultaneously using a fused-multiply-add vector intrinsic, and written back to the image with an unaligned store intrinsic combination. This optimisation reduced render kernel computing times up to 10% dependent on the scenes rendered, and the testing indicated the overhead of loading and unloading vector registers was only outweighed by processing 5 consecutive pixels simultaneously, rather than any smaller number.

The experience of trying to push the vectorization capabilities of the compiler and investigating different areas of the algorithm in an effort to optimise for vectorization has led to the recommendation that while the compiler can be very useful in automatically vectorizing code, it is still possible to gain significant performance boosts for complex kernels by manually inserting intrinsics.

3.3.3 Tuning

A set of tunable parameters are a common result of the implementation and optimisation of an algorithm. Such parameters may be used to tune algorithm behaviour for different hardware, or to tune hardware/environment behaviour to better suit the algorithm. A small number of parameters were empirically tuned to improve performance for Splotch on the Xeon Phi. Rendering parameters, such as the size of the tile used for image decomposition, can be modified via a parameter file passed in at runtime. As such, these may be tuned via a series of scripted tests iterating through incremental sets of reasonable values, defined heuristically based on the

```
#!/bin/bash
#SBATCH –nodes=1
#SBATCH –tasks=1
#SBATCH –time=01:00:00
for i in [parametervalues]
do
   echo log_file=log_$i
   echo parameter_name=$i > temp
   cat default_params.par temp » current_test.par
   srun -N 1 ./executable current_test.par &» log_file
done
```

FIGURE 3.7: A SLURM batch submission job script for iterating through a list of potential values for a tunable parameter. Performances are extracted from the log file and plotted to determine which value provides best performance.

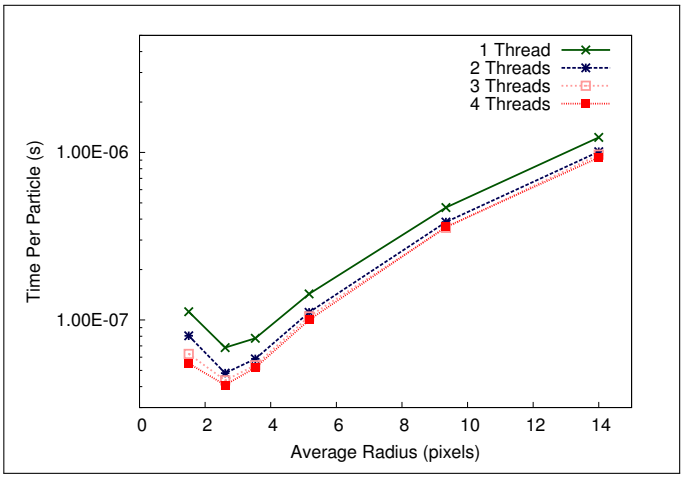


FIGURE 3.8: Comparison of performance with one to four threads per core, using best observed *scatter* affinity settings.

Xeon Phi architecture. A simple process was used in this case, using an iterative test script and extracting performance from the log file for each test, as shown in Figure 3.7. In the case that the parameter space is large, such an approach would not be practical, and alternatively an autotuning approach may be required. Autotuning is a popular approach in the field of high performance computing, for a comprehensive overview see Balaprakash et al. (2018).

Thread count and affinity are important factors in the tuning process. The effect of varying the number of threads per core and thread affinity is examined for a series of Splotch renderings. A set of tests are employed varying the camera position in order to have a fair comparison of the effect as a function of the average particle radius (see Figure 3.8). One to four threads per core are tested, four being the maximum number of hardware contexts available per core. A series of preliminary tests indicated the *scatter* affinity[8] is ideal for this use case and so this configuration is

[8]For more on thread affinity see: https://software.intel.com/en-us/articles/openmp-thread-affinity-control

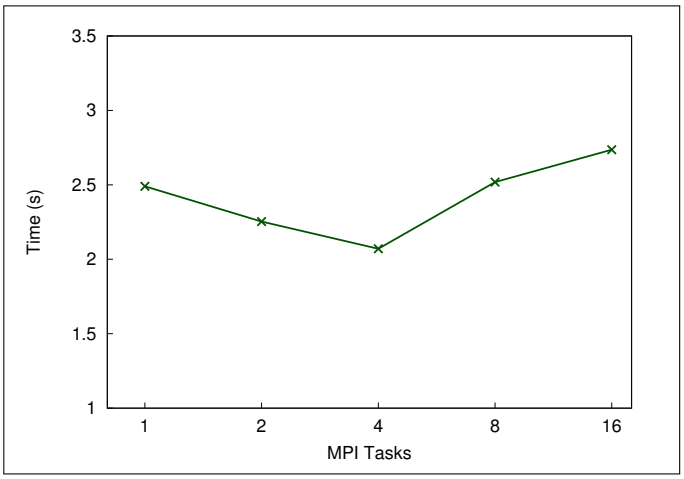


FIGURE 3.9: Per-Frame processing time comparing multiple MPI tasks on the host sharing a single Xeon Phi; each task is further parallelised through OpenMP to use the full number of hardware thread contexts available.

used for the final tests, although it is noted that in the case of 4 threads per core the difference between affinity settings (in particular *scatter* and *balanced*) was negligible most of the time, as also mentioned in Reid and Bethune (2014).

From Figure 3.8 it can be seen that the best performance is obtained for 4 threads per core. It is important to run these tests, which can be trivially scripted, for any multi-threaded application as it is likely a suitable configuration will be different from that found here. It can also be noted that, as expected, the gap between one to two threads per core is noticeably larger than between two to three and three to four; this is likely due to the inability of the core to issue two instructions from a single hardware thread context in back to back cycles (Intel, 2012f).

During tuning it was found that for relatively small datasets where processing time is low, i.e. a matter of seconds, initialization of the device and OpenMP threads can cause a noticeable overhead. The impact of this can be minimized by placing an empty offload clause with an empty OpenMP parallel section near to the beginning of the program, in order to overlap this overhead while other host activity is occurring, in this case while reading input data. Alternatively the environment variable OFFLOAD_INIT can be set to pre-initialize all available MIC devices before the program begins execution.

Beyond threads, it is also important to tune hybrid parallelism where available. Splotch supports both shared memory parallel model OpenMP, and distributed memory parallel model MPI. Typically, OpenMP is used with a node whilst MPI is used across nodes. However, even on node there are NUMA domains which can affect the performance of the OpenMP based model, and with a large number of threads on a single device there is a strong potential for workload imbalance. As such, it may be beneficial to subdivide the available device threads amongst multiple MPI processes, in an effort to minimise NUMA effects and workload imbalance.

Figure 3.9 shows comparison of per-frame processing times using varying numbers of MPI processes on the host, offloading OpenMP parallelised code to a single Xeon Phi (test data and hardware configuration is detailed in the following section). It can be seen that doubling the number of MPI processes increases the performance by 10%, up to four processes; above four it appears that the overhead of additional MPI processes causes performance to deteriorate.

3.3.4 Results

3.3.4.1 Hardware and Test Scenario

All tests were performed using the now-decommissioned Dommic facility at the Swiss National Supercomputing Centre, Lugano. In this eight node cluster, each individual node is based upon a dual socket eight-core Intel Xeon 2670 processor architecture running at 2.6 GHz with 32 GB of main system memory. Two Xeon Phi 5110 MIC coprocessors are available per node, making up to sixteen *Knights Corner* coprocessors available.

The sample dataset used for measurements is *snap092*, a snapshot of an N-Body SPH simulation performed using the Gadget code (Springel, 2005), kindly provided by Klaus Dolag of the Max Planck Institute for Astrophysics, Garching, Germany, and used throughout this chapter. For the single node animation tests, this is filtered to 50 million gas particles and 5 million star particles ($\sim$1.8 GB) in order to process a single chunk of data and more accurately measure individual kernels over many animation frames, whereas for all other tests the full size of 200 million gas particles and 20 million star particles ($\sim$7.2 GB) is used (Figure 3.10).

3.3.4.2 MIC Performance

A 100 frame animation with the camera orbiting the data is used to measure average per-frame timings producing images of 1024^2 pixels. For performance comparisons the device uses a tile size parameter of 40 pixels, a heuristically chosen best-observed value; due to cache sharing with particle data, varying this parameter within reasonable range (i.e. more 'natural' choices such as 32 or 64 pixels) has little effect on performance.

For clarity, one thread per core is used for all OpenMP tests on the Xeon. For MPI offloading to device, each task is allocated an even share of the 236 hardware thread contexts on the device which are then exploited with OpenMP, and in this case thread binding is set explicitly. In the case of one task using the whole device, thread affinity is set as per the best-observed settings for this use case (see Section 3.3.3).

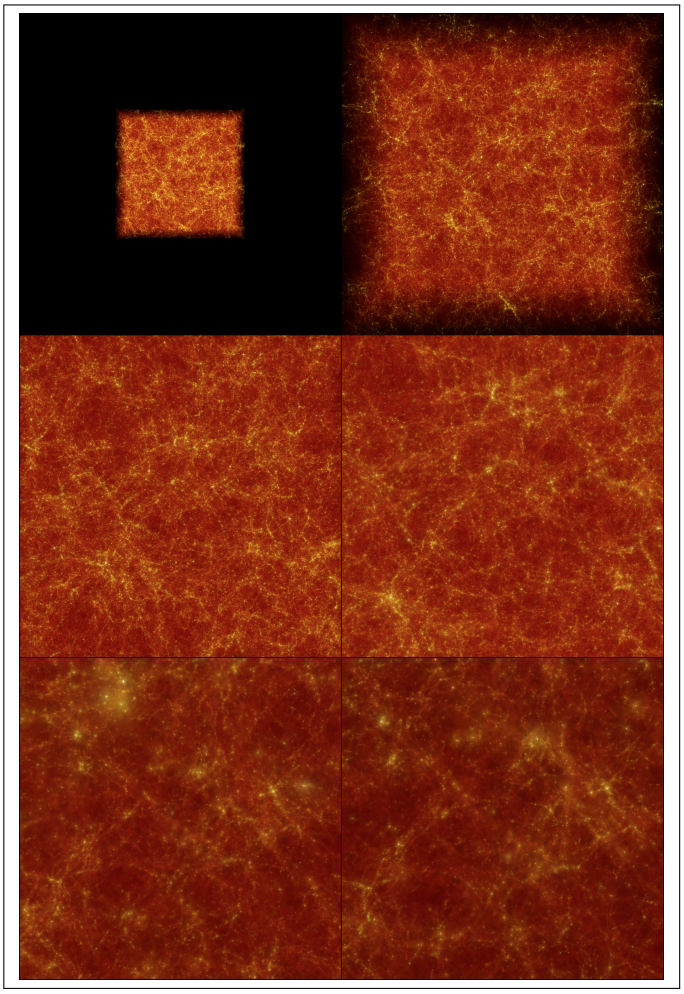


FIGURE 3.10: Six Splotch images of the dataset used for performance testing, seen from the most remote distance (*top left*) and closest distance (*bottom right*). The images depict *snap068*, a snapshot of an N-Body SPH simulation containing 200 million gas particles and 20 million star particles, performed using the Gadget code (Springel, 2005), kindly provided by Klaus Dolag of the Max Planck Institute for Astrophysics, Garching, Germany.

Performance I: Single Node Speed-up

Performance is compared for the OpenMP parallel version of Splotch on the Xeon and the Xeon Phi implementation exploiting both OpenMP and MPI. Figure 3.11, describing per frame processing times of the OpenMP Xeon implementation vs dual and single devices, shows that use of a single device provides results close to 16 threads on the Xeon. Figure 3.12 shows the strongest area of improvement, the rasterisation phase, with a single device outperforming 16 threads (two CPUs) roughly 9x, with roughly 18x improvement provided by using dual devices. In both cases the use of a second device provides a 2x performance improvement for the MIC algorithm. The best performance in terms of FLOP/s is achieved in the transform kernel, which roto-translates, projects and filters particles as a subset of the rasterisation phase. PAPI measurements indicate this kernel in a single device achieves ∼300 GFlop/s, or 15% of peak S.P. performance.

Performance II: Multiple Node Scalability

In Figure 3.13 three sets of tests are employed running from serial to highly parallel with multiple paradigms using a dual socket 16 core node; 1 to 16 OpenMP threads (OMP), 1 to 16 single threaded MPI tasks with one task per core (MPI), and 1 to 16 MPI tasks with one task per CPU and 8 OpenMP threads per task (MPI+OMP). These are compared with a final set with up to 16 Xeon Phi (MIC).

It can be seen that for the full Splotch code performance is currently achieved with one Xeon Phi roughly similar to one CPU parallelised with OpenMP. The use of a dataset larger than device memory causes an expected decrease in performance in comparison to that shown in the single node tests, due to the additional overhead of splitting the data into chunks that will fit into device memory and transferring these chunks to and from the device. The non-linear scaling for the Xeon OpenMP

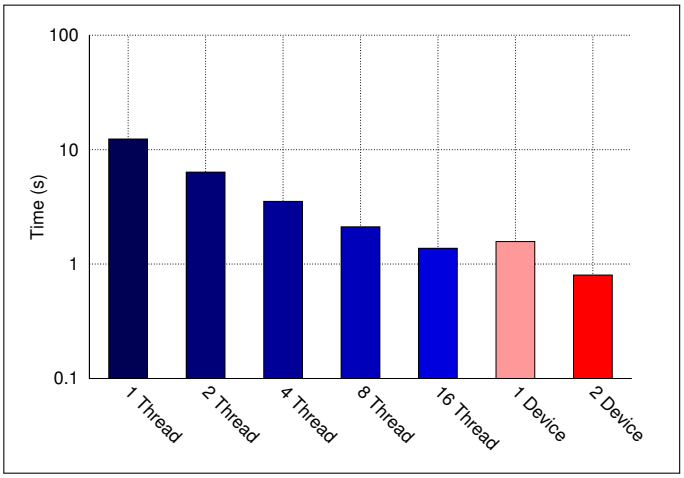


FIGURE 3.11: Per-Frame total processing time: Xeon OpenMP 1-16 threads vs single and dual Xeon Phi devices.

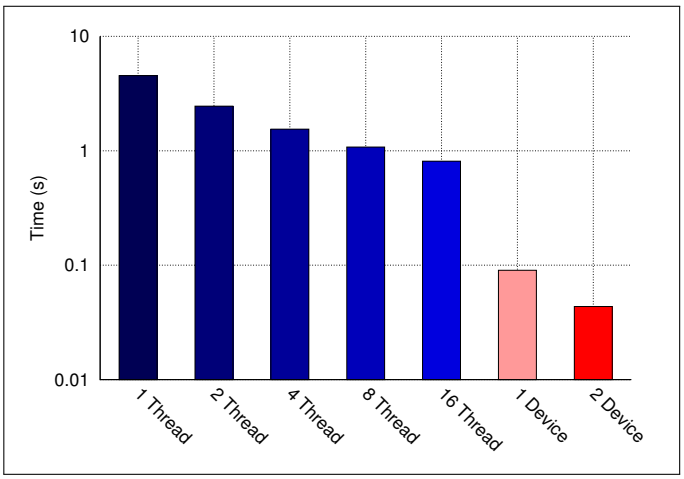


FIGURE 3.12: Per-Frame rasterisation time: Xeon OpenMP 1-16 threads vs single and dual Xeon Phi devices.

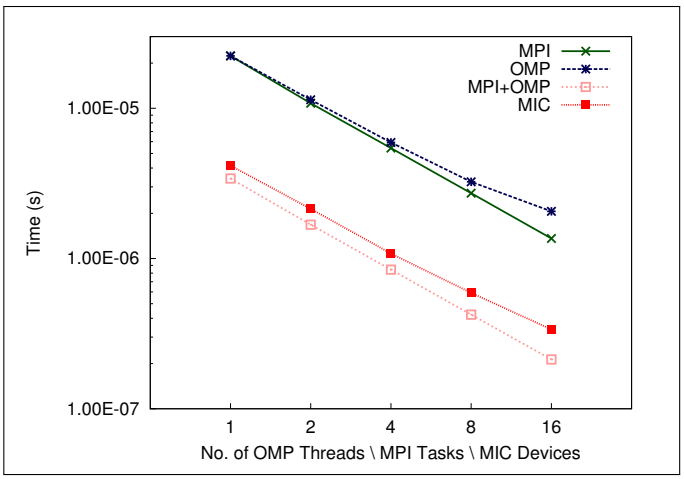


FIGURE 3.13: Scalability: Per-Particle processing time for varying models from serial to highly parallel and Xeon Phi.

implementation is due to locality issues during rendering, as threads access particles according to their position when projected onto an image rather than their location in memory. This is not an issue for the MPI implementation as each task renders particles independently of the other tasks, and there is no risk of non-local memory access. Scalability of the MIC is non-linear in the 8 to 16 range, this is due to the dataset not being large enough to fully exploit the power of the device when subdivided, more linear scaling is expected using larger datasets with device counts ranging above 8.

3.3.5 Discussion

The results gathered so far demonstrate that in some areas of code the MIC architecture excels well beyond the host capabilities, although in others a fair amount of modification is necessary to gain acceptable performance levels, which is expected of a highly parallel architecture such as this. Through the optimisation processes presented here, the code performance on Knights Corner is increased up to 6x (Fig. 3.6). Figure 3.11 demonstrates that performance equivalent to a single socket can be achieved despite the need to offload data, even for algorithms that have sychronisation challenges such as the render kernel in Splotch. As such, from a performance portability point of view it can be concluded that the optimisation process was a success, allowing the code to run with equivalent performance on a massively parallel architecture. However, as illustrated in the previous section, even in the best performance case only 15% of peak S.P. performance is reached, which indicates there is the potential to greatly outperform a traditional CPU for kernels amenable to high parallelisation. It is expected that achieving such performance will rely strongly on architecture specific optimisation; in general is recommended to make extensive use of the optimisation guides provided by Intel, and in order to achieve best performance rely not only on automatic vectorization but manual insertion of intrinsics also. Memory management is also key to performance; use of MPI based offload is shown to mitigate some overheads, similar to others' experience (e.g. (Borovska and Ivanova, 2014)). Issues regarding scalable memory allocation, which may not be apparent with an identical implementation on a Xeon CPU, can be greatly improved by use of a thread-aware allocation mechanism.

This work has focused purely on the offload model for Knights Corner models of Xeon Phi, however the optimisations performed here can be effective not only for offload processing, but also to native processing. Following this work, the second generation Xeon Phi product codenamed *Knights Landing* (Hazra, 2013) has been released and forms the basis of two of the top ten HPC systems in the world (as of June 2018[9]). A further step would be to implement the optimisations applied here to the non-offloading Splotch algorithm for native execution. The improvements to the architecture in the second generation will remove many barriers to performance;

[9] https://www.top500.org/lists/2018/06/

the ability to function as a standalone processor with direct access to large memory will remove costly PCIe based data transfer, along with providing opportunities to explore the benefits of MCDRAM high bandwidth memory. The move to Atom based cores will allow for more advanced architectural features to be exploited, e.g. out of order execution, providing improved serial performance. Chapter 4 shows some initial performance results utilising this hardware.

Furthermore, as discussed by Hwu, Keutzer, and Mattson (2008), the concurrency challenge applies to both the many-core paradigm and the continuing multi-core paradigm of hardware design. This is illustrated by the common characteristics seen in both many-core and multi-core processors; for example, multi-core Intel Skylake SP processors also support the 512-bit wide vector instruction set of the Intel Xeon Phi (Colefax Research, 2017), and so vectorization-based optimisation techniques presented here will likely also be well suited for Intel Skylake SP processors. These common characteristics increase the transferability of this work beyond the specific model and architecture of the Intel Xeon Phi, with applicability to both multi-core processors and future architectures.

3.4 Accelerators Part II: GPU

Further to the addition of new accelerators such as the Intel Xeon Phi, existing accelerators are improving with each new architecture released. In order to maintain performance on evolving architectures, algorithms and implementations should be revisited to take advantage of new features that may require explicit utilisation to improve performance. In this section, the GPU port of Splotch is re-visited to explore the utility of atomic operations on NVIDIA Kepler and later architectures.

An general understanding of NVIDIA GPU hardware and CUDA is assumed, a thorough outline can be found in the latest NVIDIA architectural white papers and programming guides e.g. (NVIDIA Corporation, 2017a; NVIDIA Corporation, 2018b).

3.4.1 Splotch on the GPU

The initial GPU port of Splotch is presented by Rivi et al. (2014). The authors develop a CUDA-based tiling schema similar to the OpenMP-based Splotch algorithm for CPUs, however experience particular difficulties in optimising concurrent rendering of large particles, whose radius depends both on the source data and the camera point of view. The overlapping nature of such particles means that care must be taken to avoid race conditions when updating image pixels during rendering. The original CPU-based algorithm relies on the relatively high performance of a single thread; image tiles are distributed to threads and each thread process all particles affecting a single image tile. This approach is not sufficient for the GPU, which requires more fine-grained thread parallelism to efficiently exploit the *single*

instruction multiple thread (SIMT) paradigm employed by CUDA thread blocks (or *warps*).

The authors develop a particle classification and hybrid rendering technique, allowing small particles that affect few pixels to be rendered in parallel on the GPU while large particles affecting many pixels are returned to the host. The classification scheme is summarised in Figure 3.14, labeling particles that affect a single pixel as *small*, those that have a radius smaller than the tile width as *medium*, and those with radii larger than the tile width as *large*. Small particle contributions are calculated with a single thread per particle approach, and then added to pixels via a parallel reduction. Medium particles are sorted by tile, and then all particles for a specific tile are rendered sequentially by a CUDA warp using a single thread per pixel and tile halo approach. Large particles are copied back to the host for rendering. A pseudocode description of the hybrid rendering approach is summarised in Figure 3.15, exploiting the *thrust* library for CUDA operations to sort particles, before handling each category as outlined. Figures 3.14 and 3.15 are replicated with permission from Rivi et al. (2014).

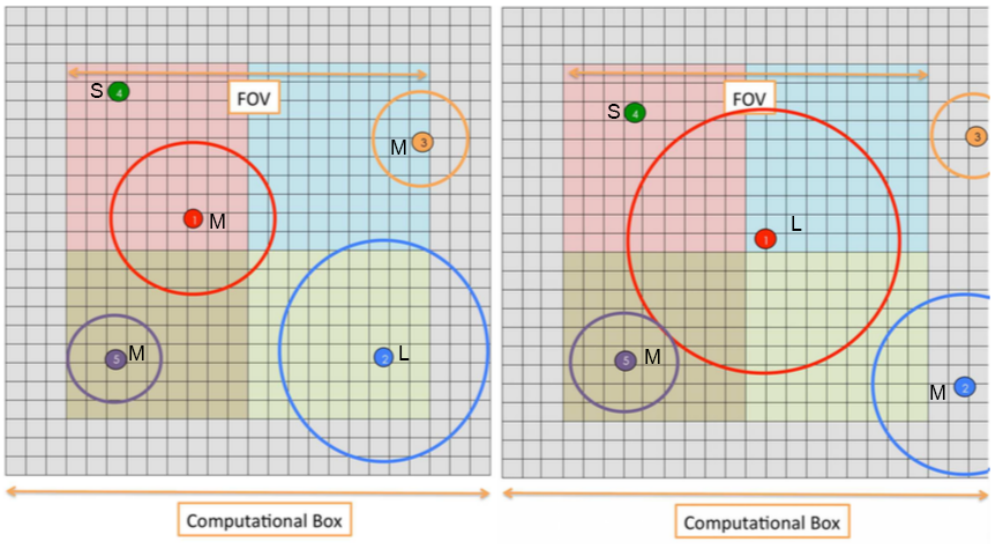


FIGURE 3.14: The classification scheme for tiled particle rendering, replicated with permission from Rivi et al. (2014). The view of five particles projected to image pixels is represented from two camera viewpoints, *left* and *right*. Tiles are represented by coloured regions of pixels. Each particle is numbered, and labelled with the classification S, M, L for *small*, *medium*, and *large*.

3.4.2 An Atomic GPU rendering approach

As discussed in earlier sections of the chapter, hardware vendors are continually designing and implementing architectural improvements. In the case of NVIDIA GPUs, with the introduction of the Kepler and later architectures, 'Atomic operation throughput to a common global memory address is improved by 9x to one operation per clock' (NVIDIA Corporation, 2017b). This is significantly increased as compared to the Fermi architecture used by Rivi et al. (2014), which is noted by the authors for

```
For each chunk of particles:
  1. synchronous copy of particles to the GPU global memory
  2. launch kernel for rasterisation
  3. device synchronization
  4. select large particles using thrust::copy_if
  5. if (number of large particles > 0):
       asynchronous copy of large particles to the host memory
  6. remove non-active and large particles from the device
  7. sort particles on the device, according to their tile id, using thrust::sort_by_key
  8. reduce number of particles and their starting position for each tile using
thrust::reduce_by_key and thrust::inclusive_scan
  9. if (number of small particles > 0):
       launch kernel computing small particles pixel location reduce pixels by key
       using thrust::reduce_by_key
       launch kernel to write partial image
       device synchronization
  10. launch kernel for rendering medium particles
  11. if (number of large particles > 0) :
         call rendering of large particles on the host
  12. device synchronization
  13. launch kernel to add partial images
  14. synchronous copy of the image to the host memory
  15. add host image with the device image
update final image
```

FIGURE 3.15: Pseudocode illustrating the tiled approach to rendering particles, replicated with permission from Rivi et al. (2014).

having poor atomic performance, necessitating the tiled approach described in the previous section. Consequently, a performance benefit may be seen by implementing a simpler approach that relies on fast atomic operations to synchronise writes to image pixels.

To test the performance of atomic operations for particle rendering, the tiled and hybrid rendering algorithm described in Figure 3.15 is reduced to a simpler algorithm described in Figure 3.16. The classification step, sorting, and multiple rendering kernels are removed in favour of a single render kernel adopting a single thread per particle approach for all particles. In this kernel, writes are implemented using the CUDA *atomicAdd()* function, to avoid data races when updating pixels. This new rendering implementation is enabled in Splotch via a compile time option; to gauge the effect of the atomic GPU rendering approach, a separate build of Splotch is compiled for each implementation and a series of performance tests are run that replicate those shown in Rivi et al. (2014). The approaches are then compared with each other and the CPU algorithm as a baseline, with further comparison to the Xeon Phi implementation in Section 3.5.

For testing, the utilised hardware consists of an NVIDIA Tesla K20X, based upon the Kepler architecture, and an 8-core Intel Sandybridge Xeon E5-2670 CPU. This

For each chunk of particles:
1. synchronous copy of particles to the GPU global memory
2. launch kernel for rasterisation
3. device synchronization
4. launch kernel for rendering all particles
5. synchronous copy of the image to the host memory

update final image

FIGURE 3.16: Pseudocode illustrating the atomic approach to rendering particles, without the classification and tiling schemas of Rivi et al. (2014).

particular graphics hardware was chosen as the latest available, at time of writing, that includes the improved atomic operation performance as detailed in Section 3.4.2. Two test datasets are employed. The first consists of a mock galaxy catalogue produced from MICE cosmological simulations (Carretero et al., 2015), kindly provided by Pablo Fosalba, IEEC-CSIC, Barcelona. The MICE catalogue data consists of ∼500M galaxies or ∼18GB storage. The image size rendered for this data is 800^2 pixels, a typical image size used in Splotch. The second test dataset *snap068*, as used previously, rendering images of 1024^2 pixels to match those of Rivi et al. (2014) (six of which are shown in Figure 3.10).

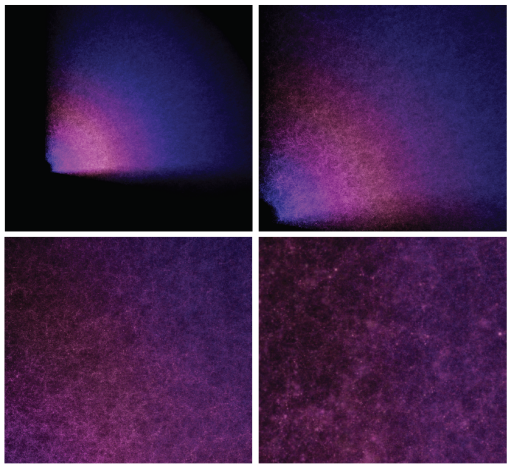


FIGURE 3.17: A sample of the test images showing a galaxy catalogue zoom-in.

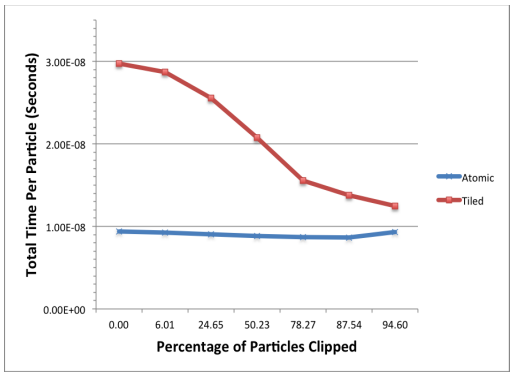


FIGURE 3.18: Total time per particle plotted against percentage of data clipped.

The atomic and tiled implementations are first compared by rendering a series of frames, a selection of which are shown in Figure 3.17, zooming in to the MICE catalogue data. In this comparison most particles affect a small number of pixels (no more than 4), and the atomic update performs very well against the tiled schema, due to avoiding the overhead for sorting and tiling particles. Total time per particle is shown in Figure 3.18. This plot shows a max speed-up of 3.1x where all particles are visible, dropping to 1.3x where many particles are clipped. The smaller range of

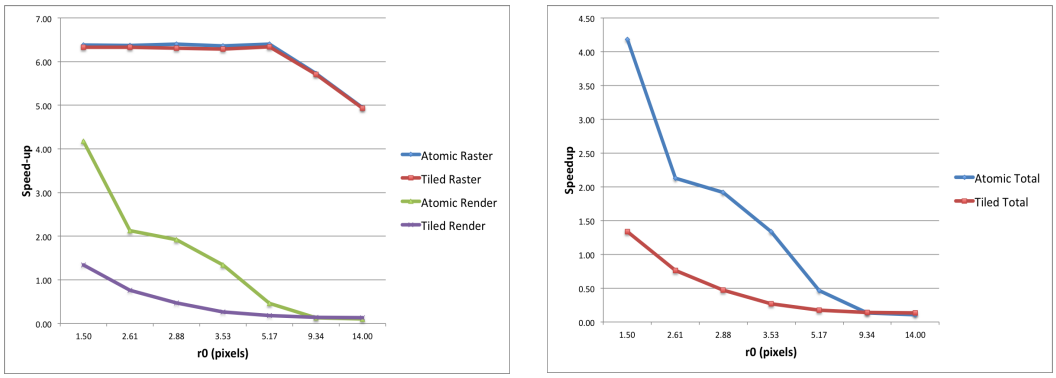


FIGURE 3.19: Speedup against 8-core CPU (*left*: single kernels, *right*: total computation).

speed-up is due to decreased overhead in the particle classification and tiled rendering stages as many particles are clipped early-on in the algorithm.

A further comparison in Figure 3.19 gauges speed-up in both total compute time and individual kernels against the CPU algorithm parallelised with OpenMP and running on 8 cores of a Sandybridge CPU. The Gadget N-body simulation dataset used in this case is characterised by many overlapping particles with large radii (large r0) at close range; this can be particularly difficult for parallel rendering as described previously. The rasterisation is highly parallel at between 5x and 6x speedup, this is unchanged from the previous implementation. For smaller radii (lower r0), the atomic rendering code provides speed-up over the CPU in four out of seven cases, in comparison to a single case for the tiled code.

3.4.3 Discussion

The implementation used in (Dykes, Gheller, Krokos, et al., 2015) takes advantage of improved atomic operations in NVIDIA GPUs of Kepler architecture and beyond, replacing the tiling and classification schema with a simpler strategy making use of the CUDA atomic add function to safely accumulate overlapping particles to a single pixel in a parallel context. The result is both a reduction in code complexity, and an improvement in algorithm performance.

However, there is still potential for further performance gain. For example, the new implementation approach does not consider the GPU memory hierarchy, and it is likely that performance is negatively impacted by randomly accessing pixels in global memory. Informed use of cache and shared memory can have strong effects on performance and require very specific code tuning (e.g. (Lobeiras, Amor, and Doallo, 2012), (Baghsorkhi et al., 2012)), therefore it is likely that an approach introducing a form of tiling similar to (Rivi et al., 2014) but focused on retaining data in shared memory while exploiting atomic operations will provide further performance improvements.

Furthermore, for a GPU to be exploited, a CPU is generally required to act as a host. In the more general case of GPGPU, the ideal scenario is that the CPU offloads

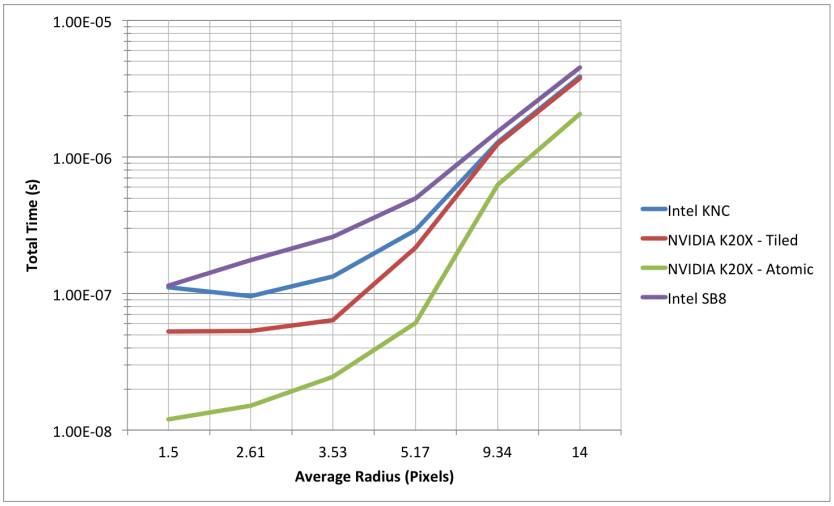


FIGURE 3.20: Intel KNC Xeon Phi vs. NVIDIA K20X Atomic and Tiled vs. Xeon 8c Sandy Bridge CPU: comparison of the total compute times on a per-particle basis.

data and operations that are well-suited to the GPU and continues processing of its own data such that both processors are saturated with work. In the current scenario, all data is offloaded to the GPU and the CPU acts predominantly as a coordinator. A further step in the performance investigation should consider splitting the data and offloading a portion to the GPU to ensure both processors saturated with work.

3.5 Accelerators Part III: A Comparison

The experience of implementing the Splotch code for both GPUs and Xeon Phi allows to make a comparison of the performance achieved through similar expenditures of time and effort. In the case of the GPU the algorithm was implemented in CUDA as detailed in Section 3.4, with both tiled and atomic schemes tested with matching hardware to Section 3.4 (NVIDIA K20X). In the case of the Xeon Phi the optimised parallel model with OpenMP was tested with matching hardware to Section 3.3 (Xeon Phi KNC 5110). The same full dataset and host processors are used for performance tests. An image of 1024^2 pixels is rendered for six different camera positions (Figure 3.10, starting from very far and reaching progressively very close to the center of the simulation). In order to make a comparison on a per-particle basis the total compute time (i.e. full algorithm minus data read and image output), the rendering kernel time, and the rasterisation kernel time are measured. For completeness, a comparison against 8 OpenMP threads exploiting an 8-core Intel Xeon Sandy Bridge E5-2670 is included.

Figure 3.20 shows total performance of a single NVIDIA K20X card versus Xeon Phi KNC and Intel Sandy Bridge 8 core CPU, as function of the average particle radius. The average radius, while not being the only factor affecting rendering time (see (Rivi et al., 2014) for more detail), is a useful metric for comparison; a larger radius means the particle will affect more of the image and computational cost will

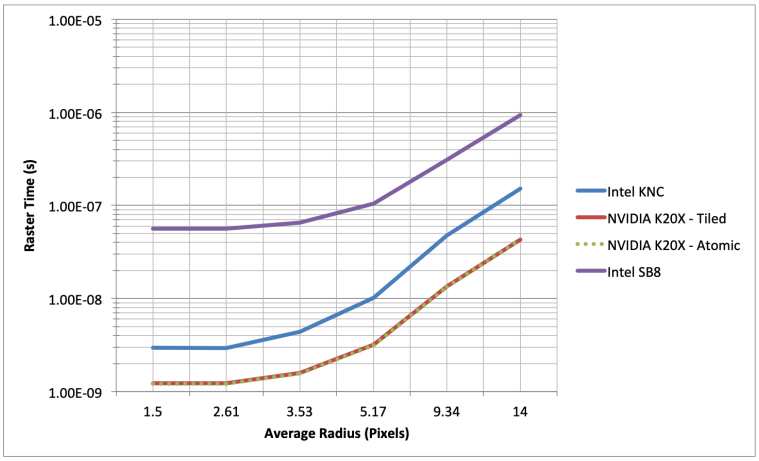


FIGURE 3.21: Intel KNC Xeon Phi vs. NVIDIA K20X Atomic and Tiled vs. Xeon 8c Sandy Bridge CPU: comparison of the rasterisation per-particle times. Note that GPU results are almost identical (overlaid on plot) as tile/atomic schemes do not affect this kernel.

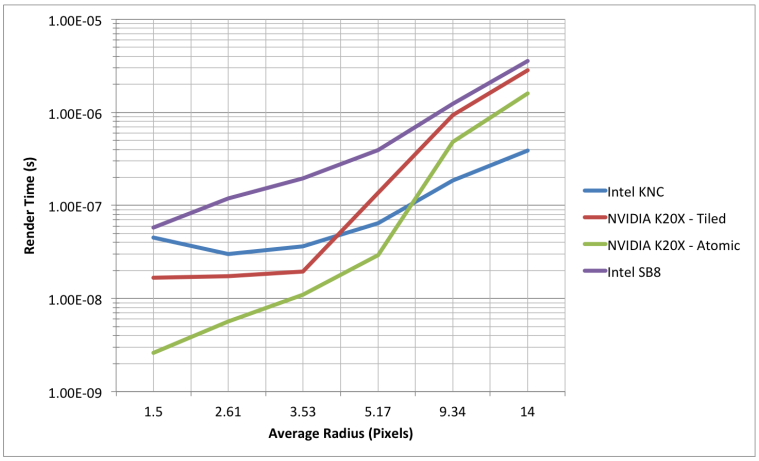


FIGURE 3.22: Intel KNC Xeon Phi vs. NVIDIA K20X Atomic and Tiled vs. Xeon 8c Sandy Bridge CPU: comparison of the rendering per-particle times.

increase. It is clear that the atomic GPU implementation outperforms the Xeon Phi in all tests, although for the larger radii the results are very similar to the tiled implementation.

Figures 3.21 and 3.22 show kernel specific performance on a per particle basis. The performance difference shown in Fig. 3.21 is mostly attributed to the combination of the colourise and transform/filter kernels on the GPU. This means that for a large dataset where a significant portion of particles are inactive (i.e. off screen), as is the case for the tests with larger average radius, the kernel ends before processing these particles. To retain automatic vectorization of the transform kernel on the MIC the colourise kernel is run separately, and so all particles must be re-read and the active status tested, causing the colourise kernel to be dependent on the total number of particles as opposed to solely the number of active particles as is the case on the GPU. Note that GPU results are almost identical (lines are overlaid) as tile vs atomic schemes do not affect these kernels.

Figure 3.22 shows performance comparison for the rendering phase. For the largest average radii, above nine pixels, the MIC outperforms the GPU for rendering. This is due to the MIC algorithm being more suited to scenarios where a particle may affect a large portion of the image, as particles can be rendered by multiple threads when affecting multiple tiles. For the tiled GPU implementation, this is not possible as each tile is rendered by different CUDA blocks, therefore when a particle affects more than one tile it must be transferred back to the host and rendered as described in Section 3.4. The atomic GPU implementation fairs better, but still does not perform as well as the KNC in this case.

The GPU performs very well in the lower radii range due to the fact that the large majority of particles are considered point-like, and can be rendered in an efficient one thread per particle manner. The MIC performance decreases in the case where a considerable portion of the image is unused (e.g. with a point of view far from the computational box center, as in Fig. 3.10 *top left*). The current decomposition method does not effectively load balance this distribution of particles and requires improvements to account for such situations.

In most cases both the MIC co-processor and GPU accelerator outperform, or perform very similar to, the Xeon 8-core comparison CPU.

3.6 Distributed Memory Particle Rendering

A further quality typical of high performance computing systems is the use of distributed memory hierarchies. Chapter 2 introduced an improved optical model for Splotch, in order to exploit independent absorption and emission coefficients on a per-particle basis. However, as noted in Section 2.3.2, this requires to render particles in back to front order as part of the radiative transfer approximation. The MPI parallel approach to Splotch uses a distributed memory particle splatting approach that relies on a simplified optical model due to the added complexities of distributed

memory. This simplified optical model allows particles to be distributed across tasks in any manner, and images from different tasks can be composited using a simple sum operator. However, this approach is based on an assumption that the ratio of emission to absorption is the same for each particle, rather than independent as demonstrated in Chapter 2. As such, the distributed memory rendering approach cannot yet support the new optical model presented in Chapter 2. The current distributed memory rendering pipeline is illustrated in Figure 3.23, showing distinct tasks each reading a portion of the data, locally rendering, and then compositing to the master task for output.

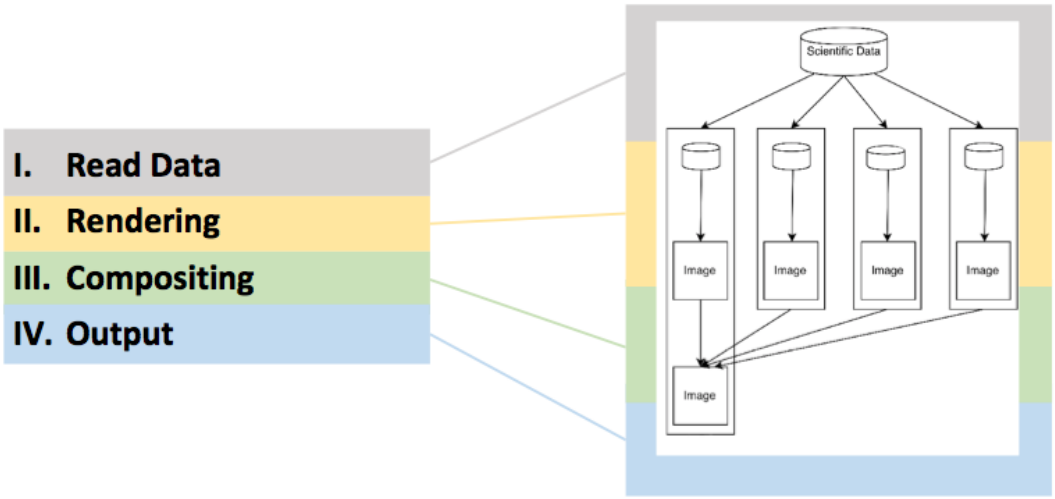


FIGURE 3.23: The current rendering pipeline for distributed memory (MPI) rendering in Splotch. Each distinct task reads a portion of the data, locally renders, and then composition is performed via parallel sum reduction to the master task for output.

There are three key difficulties to overcome to support the new optical model for distributed memory particle splatting:

1. Rendered data must be spatially coherent, to solve the radiative transfer equation locally on each rank.

2. The overlapping boundaries between spatial domains must be handled appropriately.

3. Rendered images must be composited in back-to-front order.

This section will propose an approach for distributed memory particle splatting, addressing (1) through a parallel binary tree data structure with data redistribution mechanism to sort particles into spatially coherent domains; and addressing (2) with *ghost particle* boundary handling and (3) via a sorted image compositing method. Ghost particles are a typical approach to solve boundary conditions for particle-based computational techniques, a comprehensive example can be found in the work of Colagrossi and Landrini (2003) on SPH based fluid flows.

Figure 3.24 shows the additions required to the current distributed rendering pipeline (shown in Figure 3.23), and each of the pipeline stages will be detailed in the following subsections. A series of figures showing a four task example of the parallel rendering approach will be used extensively throughout this section for illustrative purposes.

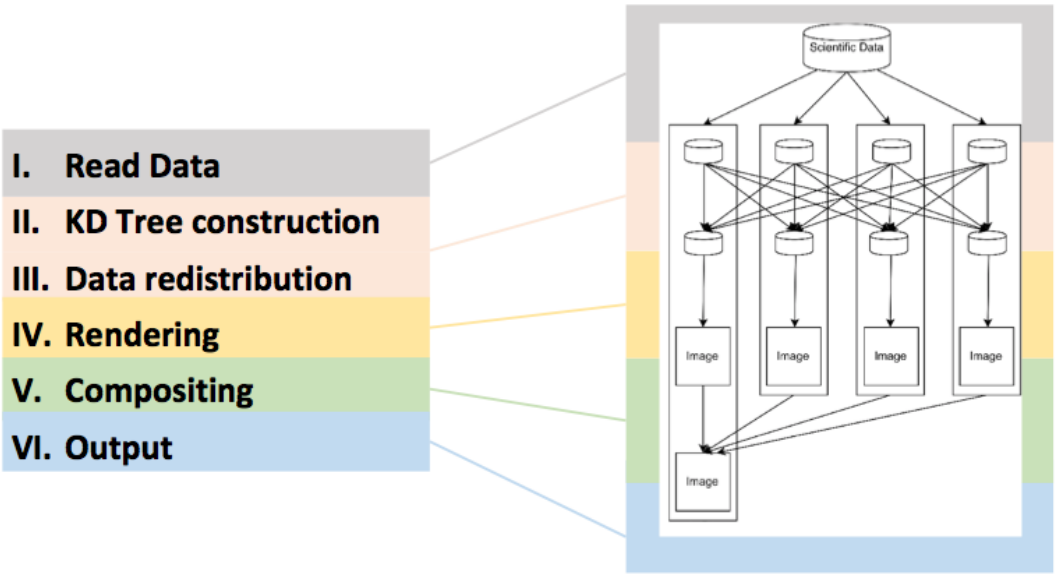


FIGURE 3.24: An extension of the distributed memory rendering pipeline for Splotch, introducing tree construction and data redistribution stages

3.6.1 Read Data

A simple parallel read utilising four MPI tasks is demonstrated in Figure 3.25, with individual particles colour coded to show in which task's local memory they reside. This stage is unmodified from the original Splotch pipeline, and highlights the spatial incoherence that can occur when loading data in parallel. This example is showing an extreme example of random distribution, and there may well be some sorting mechanism already in the dataset, however relying on this would introduce an unacceptable file-structure dependency in Splotch, so it must be assumed that data is unsorted.

3.6.2 KD Tree Construction

As highlighted in the previous section, after reading the data there is no spatial coherence. Adjacent particles may be anywhere in memory on any processor, as illustrated in Figure 3.26 (1). As such, the first step is to construct a domain decomposition that retains spatial coherency, and redistribute particles accordingly. The most important part of such a domain decomposition in this context is that domains must be *convex*. This requirement ensures that any viewing ray r may only enter and exit a domain d once, and all points p on ray segment r_s (between entry and

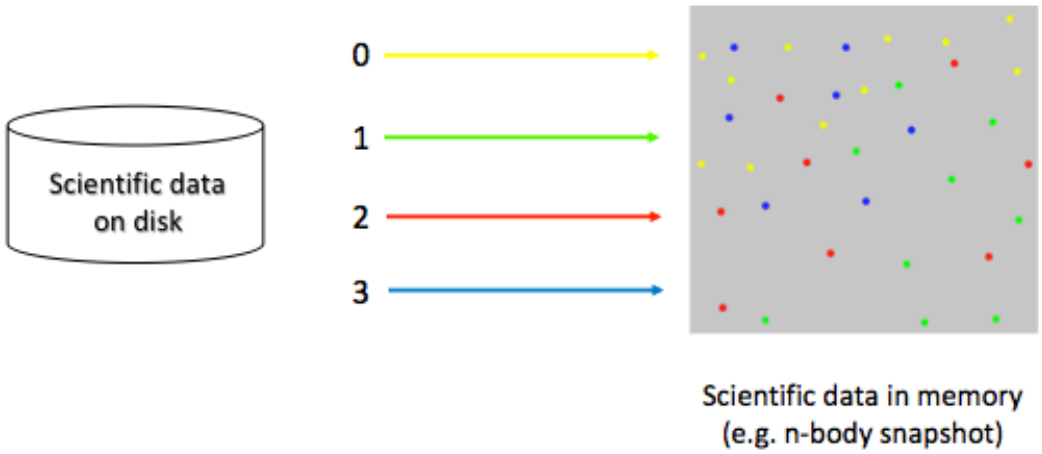


FIGURE 3.25: A simple representation of four MPI tasks reading a particle-based data file, the rank of each task is colour coded with the particles represented in the spatial domain.

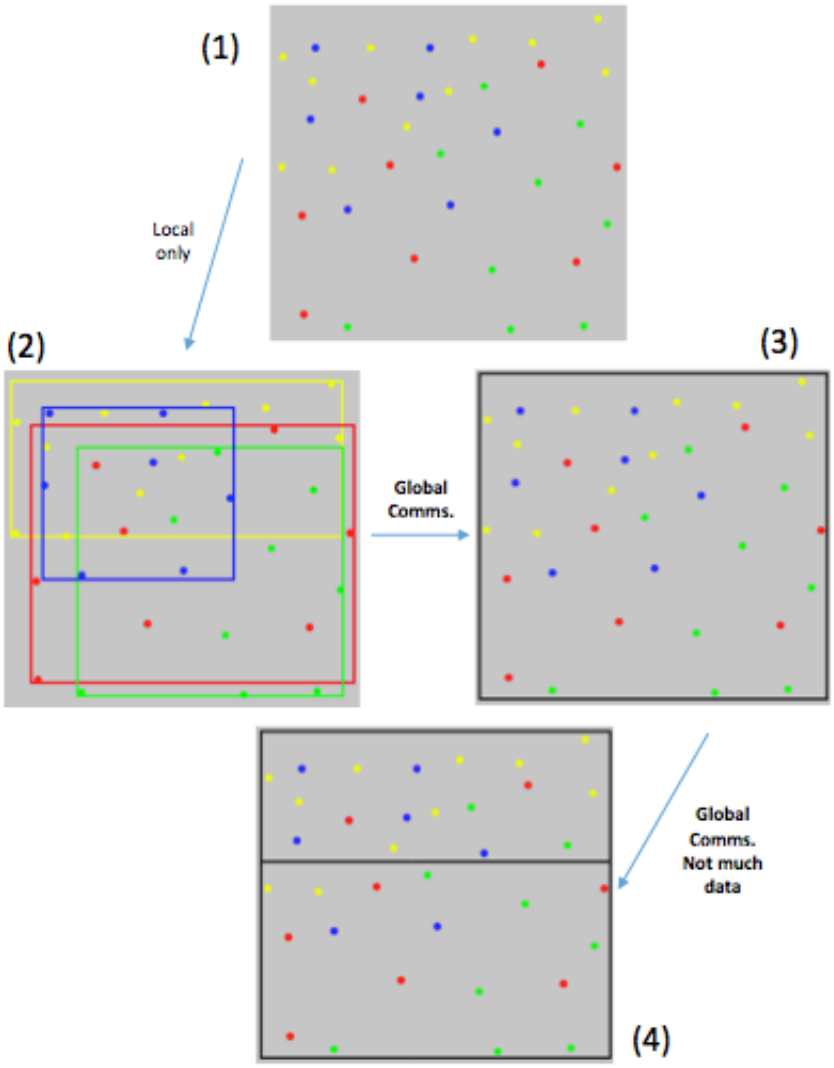


FIGURE 3.26: The four steps to begin constructing a KD binary tree in parallel. Steps 2-4 are repeated until there are enough nodes to allocated at least one to each rank. Following on from Figure 3.25, this will be four.

exit of d) are inside domain d. Such a constraint allows only one ray segment per domain which may be locally integrated (to solve the radiative transfer equation described in Chapter 2), and simplified the compositing process described in Section 3.6.5. A similar approach can be seen in, for example, the work of Leaf et al. (2013) for volume rendering adaptive mesh data.

There are a variety of existing methods for domain decomposition, as outlined and evaluated by Havran (2000), for example: Bounding Volumes with/without Hierarchy, Grid, and different types of trees (BSP tree, Octree, KDtree). Of these options, the KD tree is an effective approach in this context due to the adaptive placement of cutting planes, which is ideal for the non-regular data distributions typical of particle methods (due to their inherent adaptive resolution). Whilst typically in SPH/N-body methods, such a tree is required for optimal access to neighbouring domains, in this case the primary requirements are convex spatial domains and even data distribution.

Figure 3.26 (2-4) demonstrates the first steps of constructing a parallel KD tree. Each process calculates a local bounding box (2), and local bounding boxes on each process are reduced to a global bounding box (3). The median particle on the longest axis is chosen as a splitting plane, to divide particles into convex spatial partitions (4). Each tasks local median is found in parallel by performing a local *quickselect*, then a global median of medians is selected by collecting local medians to a master task and sorting. The global median then becomes the pivot value for a parallel *quickselect*. Steps (2-4) are repeated until enough splitting planes have been placed for each task to take ownership of at least one spatial domain.

Figure 3.27 highlights the boundary problem. During tree construction, each particle found inside a domain but overlapping a splitting plane is marked as a *remote ghost*, meaning that it is a locally stored particle but will be required as a ghost particle for remote domain (i.e. the domain(s) that it overlaps). These particles are duplicated for each overlapping domain and stored in a separate list, which will be sent to the task that owns the remote domain during redistribution. Each particle outside of a domain but overlapping the splitting plane is marked as a *local ghost*, which means it will be sent to a remote task but a local ghost copy must be retained. The axis of the identified plane is stored in a per-particle bitwise binary map for reuse during rendering.

3.6.3 Data Redistribution

Following on from tree construction in Section 3.6.2, Figure 3.28 shows the tree is constructed to a depth of four (5), with each leaf node allocated to a task. A redistribution phase swaps particles so that each task has a spatially coherent domain (6). The basic structure of the tree is illustrated in (7), each leaf node corresponding to a spatial domain and allocated to a task via colour coding.

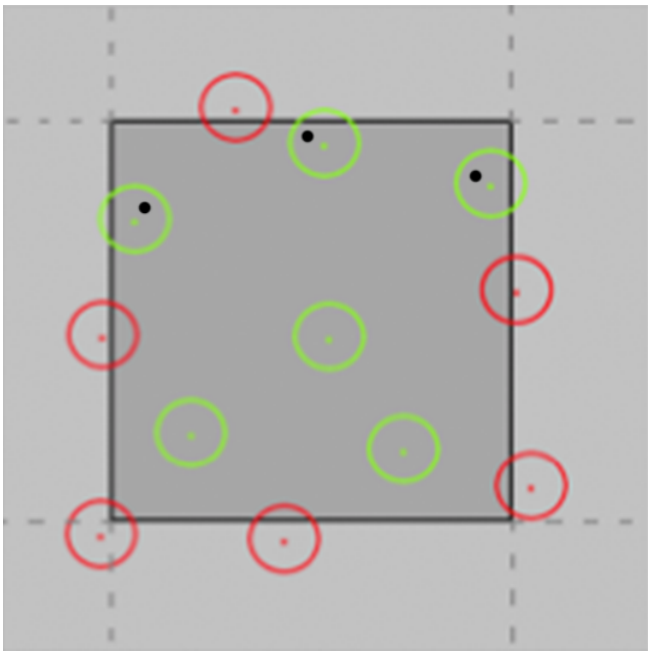


FIGURE 3.27: Particles inside the local domain are marked in green, and will be kept in local memory. Ghost particles are those that overlap domain boundaries. Particles outside the local domain are marked in red, and will be sent to remote tasks. Three particles inside the boundary are marked as *ghosts*, and duplicated to send to other tasks (green with black dot). Six particles (in red) are also marked as ghosts, and duplicated to store locally.

During redistribution, for each task all particles outside of the allocated spatial domain but stored locally are sent to other tasks, and all particles inside of the allocated spatial domain but stored on remote tasks are received, at the same time exchanging ghost particles identified during construction.

The prototype implementation uses two MPI_Allgatherv calls, once for regular data, and again for ghost data, for each tree leaf node (equivalently, for each task), however it is expected that this will not perform adequately at scale and a more advanced parallel redistribution algorithm will be required.

3.6.4 Rendering

Figure 3.29 illustrates each of the four MPI tasks locally rendering an image slice of the volume, which is approximated to a set of cubes to clearly show the convex structure and spatial coherence. Figure 3.30 highlights the domain boundary problem, where particles overlap between domain boundaries, each of the particles in red will be marked as ghosts either locally or remotely (as discussed in Section 3.6.2), and are explicitly treated during rendering.

Figure 3.31 demonstrates a *scaled cutting* approach to handling ghost particles. During rendering, if a particle is identified as a ghost (marked during tree construction in Section 3.6.2), then the contribution to a viewing ray is scaled according to

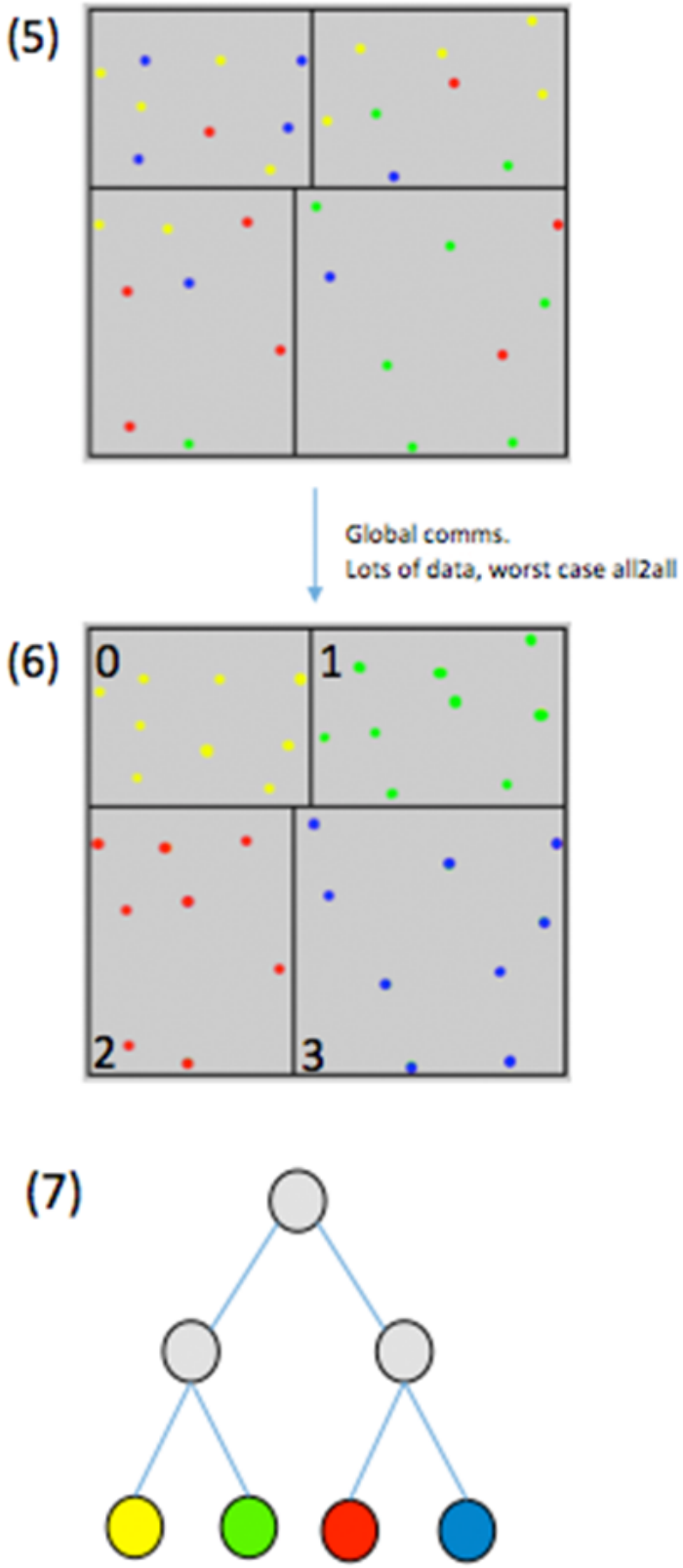


FIGURE 3.28: Following on from Figure 3.26, the tree is constructed to a depth of four, and a redistribution phase swaps particles so that each task has a spatially coherent domain.

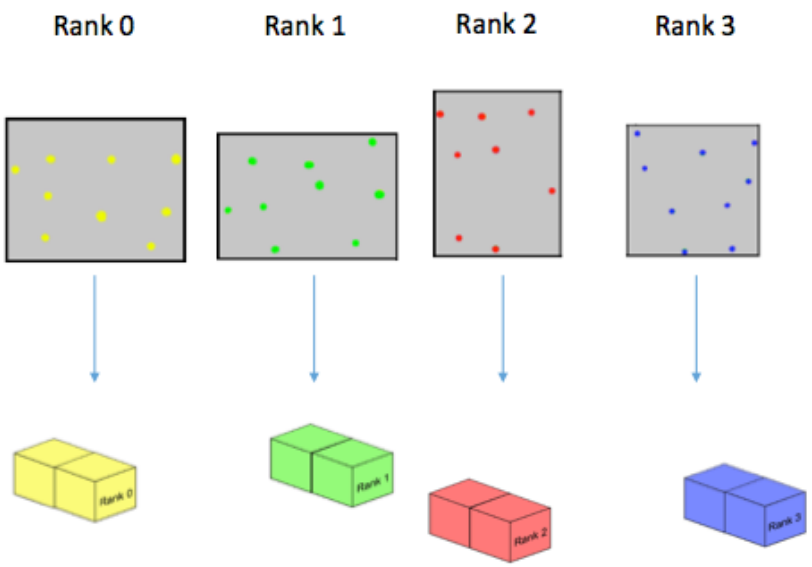


FIGURE 3.29

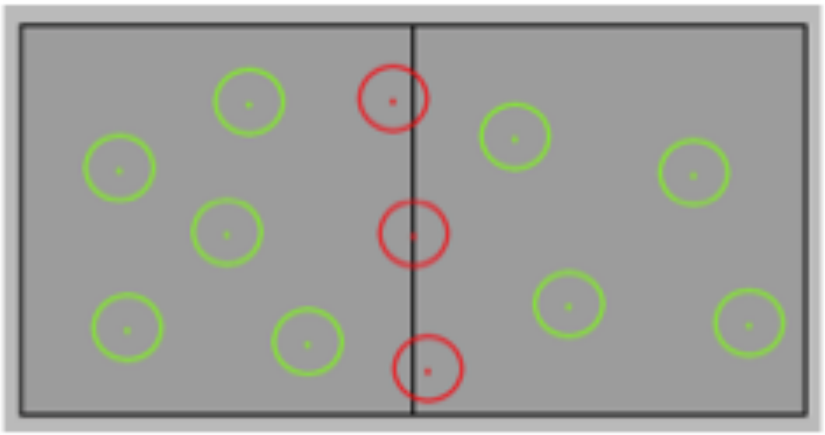


FIGURE 3.30: The domain boundary problem, each of the particles in red will be marked as ghosts either locally or remotely, and explicitly treated during rendering.

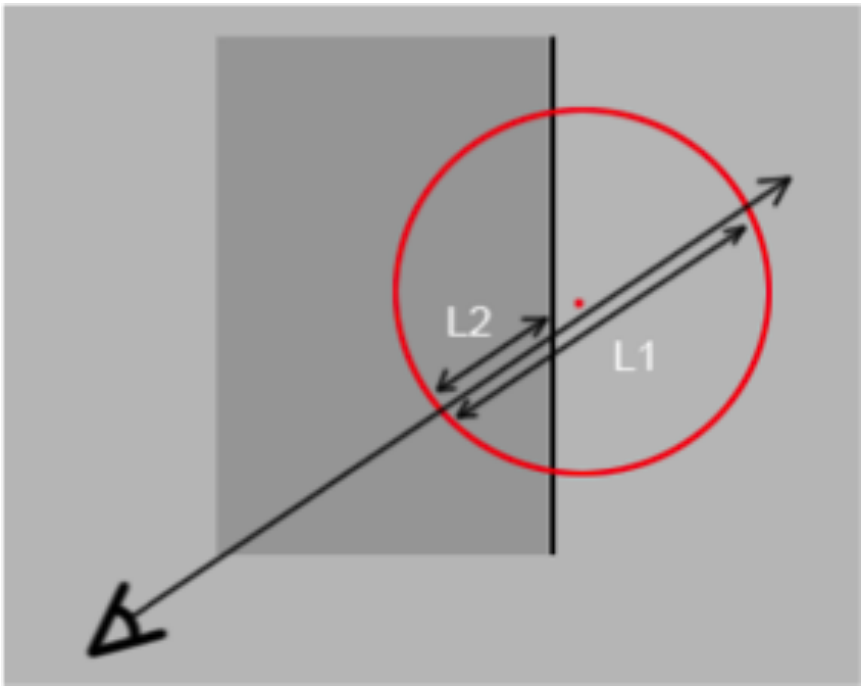


FIGURE 3.31: A scaled cutting approach to rendering boundary particles. The particle contribution is scaled by the ratio of L2 to L1 with consideration to the particles interpolation kernel weighting.

the proportion of the particle inside the domain boundary, in order to *cut* the contribution outside of the domain. An equivalent scaling is applied for the duplicated particle on the remote task, such that during compositing the particle contribution is reconstructed correctly.

3.6.5 Compositing

In order to retain the ordered integration of viewing rays, a sorted image compositing mechanism is required. Figure 3.33 illustrates the risk of incorrect ordering that may arise during composition, considering the volumes as opaque to better highlight the issue.

As such, similar to sorting particles back to front, images must be composited back to front based on the distance from the viewer. Figure 3.32 demonstrates this composition for the four rendered images of Section 3.6.4.

There are a variety of approaches to compositing, the most common of which can be summarised as:

- **Direct Send:** Each process takes 1/np of the image, all other processes send their image portions to the owner. If np is large, network can be congested with many simultaneous messages.
- **Binary Swap:** Tree based compositing method, with improvements to avoid higher tree nodes going idle as tree is traversed. This is the same as direct send if tree only has 2 members. (Ma et al., 1994)

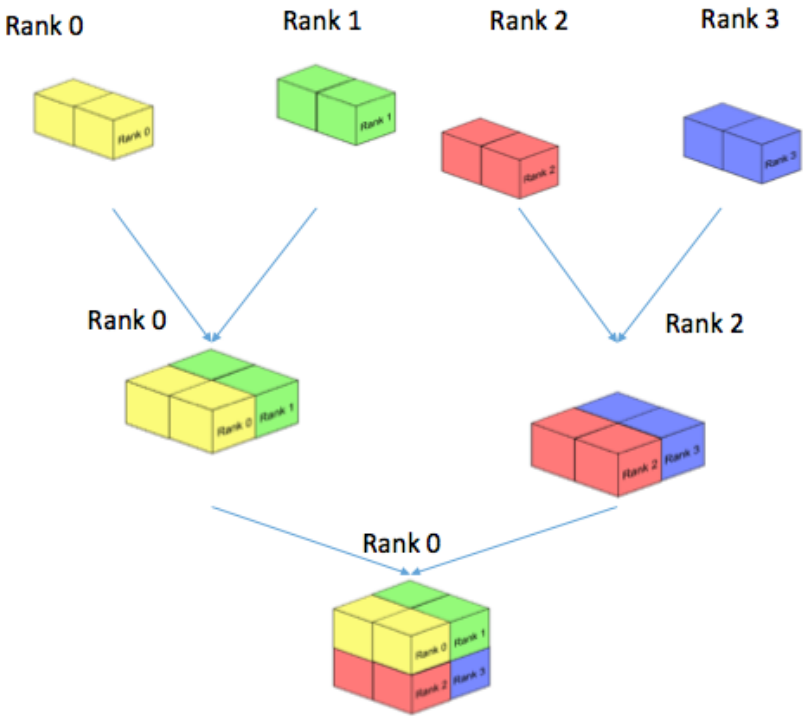


FIGURE 3.32: A 4 task parallel ordered image composition, which is implemented via the Binary Swap algorithm.

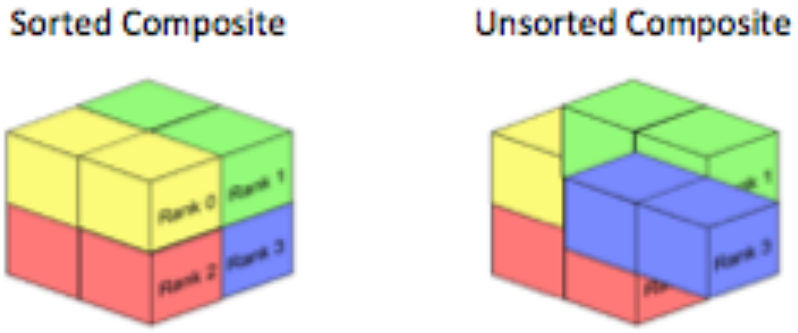


FIGURE 3.33: The potential pitfalls of incorrect compositing order, the *left* image shows a correct ordering of the composition in Section 3.32, and the *right* shows an incorrect ordering, considering the volumes as opaque for illustrative purposes.

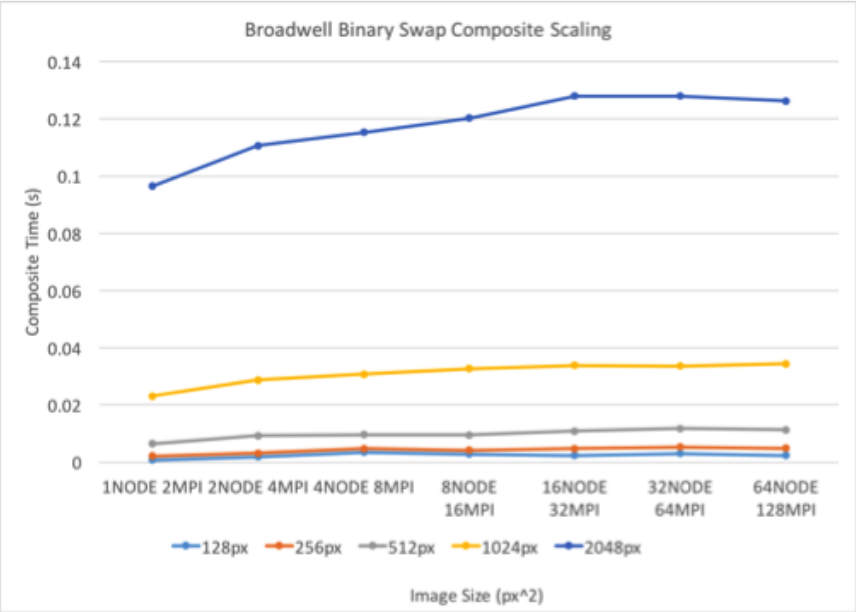


FIGURE 3.34: Scaling performance of prototype binary swap composition in Splotch.

- **2-3 Swap:** Extension of the binary swap to allow non-power2 number of processors, aiming to allow flexibility of direct send while retaining speed of binary swap. Works by allowing compositing in rounds with groups of 2s and 3s. (Yu, Wang, and Ma, 2008)

- **Radix-K:** Further generalisation of binary swap to allow k processors per group for n rounds. Direct send is still used within groups. (Peterka et al., 2009)

Furthermore, various optimization methods can be used, for example: active pixel encoding and compression, as demonstrated in the IceT image compositing library. As pointed out by Moreland et al. (2011), image compositing algorithms are considered state of the art and further research should consider optimal hardware utilisation as opposed to better compositing algorithms.

The prototype compositing stage implements a binary swap compositing methods, as a simple first step that is known to be scalable. Further work would generalise this approach to 2-3 swap, and then Radix-K. Figure 3.34 demonstrates the supported scalability of the implemented compositing algorithm, for a variety of image sizes and numbers of tasks.

3.6.6 Output

Finally the image must be output by writing to disk, this is taken care of by the master task and, like the input stage, is not modified from the original pipeline.

3.6.7 Current Status and Next Steps

This section proposes an algorithm for distributed memory particle splatting within Splotch, with a prototype implementation. Future work is envisaged to move this implementation from the prototype stage to production usage in Splotch. For production use, the parallel tree construction and data redistribution stages must be fully optimised, as well as a thorough analysis of rendering performance, which is beyond the scope of this thesis. However, a brute force approach is acceptable for prototyping, as the most time consuming tasks of parallel tree construction and data redistribution must only take place once for static datasets. Furthermore, Section 3.6.1 explains the current assumption that data is unsorted (i.e. no pre-existing spatial coherence) when read from file, to develop a technique that is not dependent on specific file formats. In future, an implementation could consider improving upon this by coupling file readers to the tree structure to exploit existing spatial coherency known in the file when possible, however must be able to revert to unsorted behaviour as default.

Currently the KD tree structure is used solely to ensure correct parallel ray integration, decomposing only deep enough to ensure each task has at least one leaf node. Future work should also consider a deeper tree structure which could support blocked rendering of nodes that can fit in GPU memory, and exploiting the structure for level-of-detail techniques such as merging contributions within tree nodes to reduce the number of calculations necessary. A key issue that will need consideration is that, for a dynamic dataset, such as visualising a simulation in-situ, this approach would need to further address updating the tree structure and redistributing data as particles move between domains.

3.7 Methodologies for Achieving Performance on Future Systems

Section 3.2 introduced some of the challenges presented by the growing trend of heterogeneous architectures in high performance computing, and identified three architectural characteristics that are expected to prevail in future systems: *increased core count*, *wider vector registers*, and *complex memory hierarchies*. The work presented in Sections 3.3, 3.4, and 3.5 demonstrates equivalent, or improved, performance of the Splotch algorithm across heterogeneous architectures by focusing on these key features. Such efforts achieved, at the very least, performance portability for the specified architectures, and in places outperformed the more traditional CPU-based systems; however, considering the wider context, there is a good indication that it is feasible to extract, generalise, and combine these methods to form methodologies that are applicable to the design and implementation of general algorithms on future architectures.

In this section, two new methodologies are presented. The first, detailed in Section 3.7.1, extracts the methods that were found effective for Splotch and are expected to be applicable to a wide range of algorithms. The second, detailed in Section 3.7.2, extracts the methods that were found effective for Splotch and are expected to be applicable to the more narrow range of high performance visualisation algorithms based on *volume splatting* (as introduced in Chapter 2). To provide a framework on which the methodologies will be built, a general three stage process of algorithm porting will be defined as follows, based on the experiences in Sections 3.3 and 3.4:

1. Algorithm (Re-)Implementation

   The stage refers to the implementation of the algorithm for a massively parallel hardware platform. This may require algorithmic re-design and/or implementation with a required programming model. The result of this stage is typically a working implementation on a specific hardware platform.

2. Optimisation

   This stage refers to the process of modifying the implementation to achieve better performance on a specific platform, taking advantage of architectural features and taking detailed performance measurements.

3. Tuning

   This stage refers to the fine-tuning of algorithmic or system parameters to find the appropriate parameter set for best performance.

The following methodologies will be framed as a set of methods to include in this three stage process, to complement typical porting and optimisation methods.

3.7.1 General Algorithms on Future Architectures

Figure 3.35 presents a methodology to be applied when porting or optimising general algorithms on future architectures, following the expected trend of features presented on the left side of the figure. In the center box of Figure 3.35, the methods found to be effective for Splotch and expected to be more generally applicable are identified, extracted, and grouped by the appropriate stage of the porting and optimisation process (as followed for Splotch); each of the methods will be briefly described.

Increased task parallelism: Increasing task parallelism requires algorithmic redesign in order to take advantage of a much larger number of cores than traditional CPU architectures. This involves identifying work that is better suited to highly parallel devices and work that is not, and aiming to make the latter type of work more suited to task parallelism - for example through advanced domain decomposition or data and task synchronisation methods such as atomic operations.

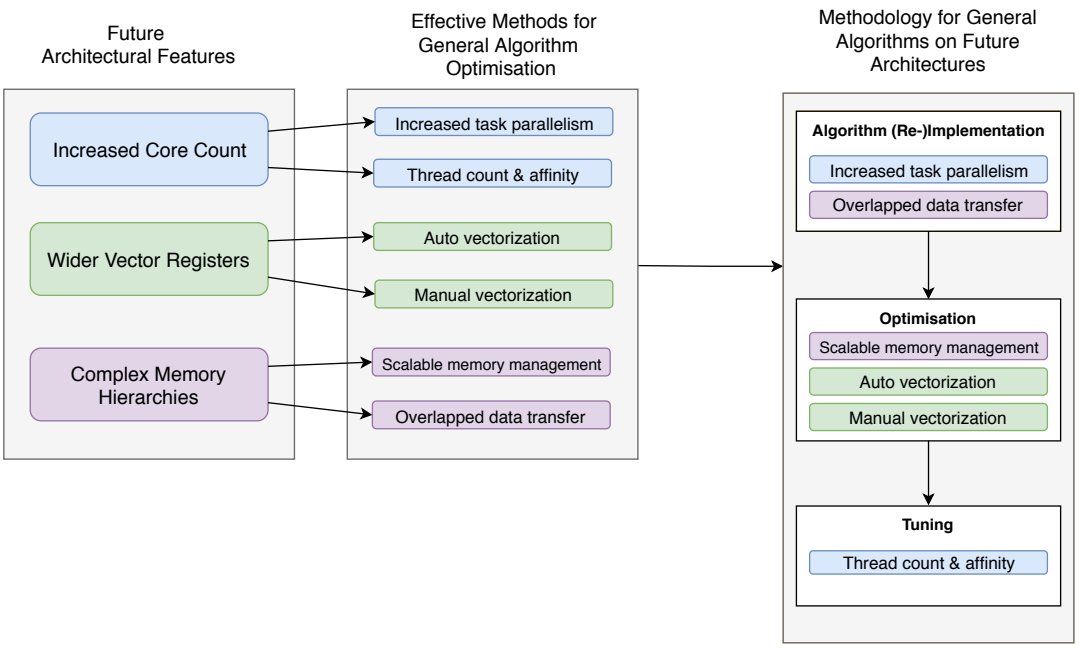


FIGURE 3.35: A methodology for general algorithms on future architectures is presented. First a relation is defined between the identified future architectural features (figure left) and the generalised methods found effective for Splotch (figure middle). These methods are then grouped by the stage of the porting and optimisation process they are most relevent to, and ordered to form a methodology (figure right) that may be followed by future implementors.

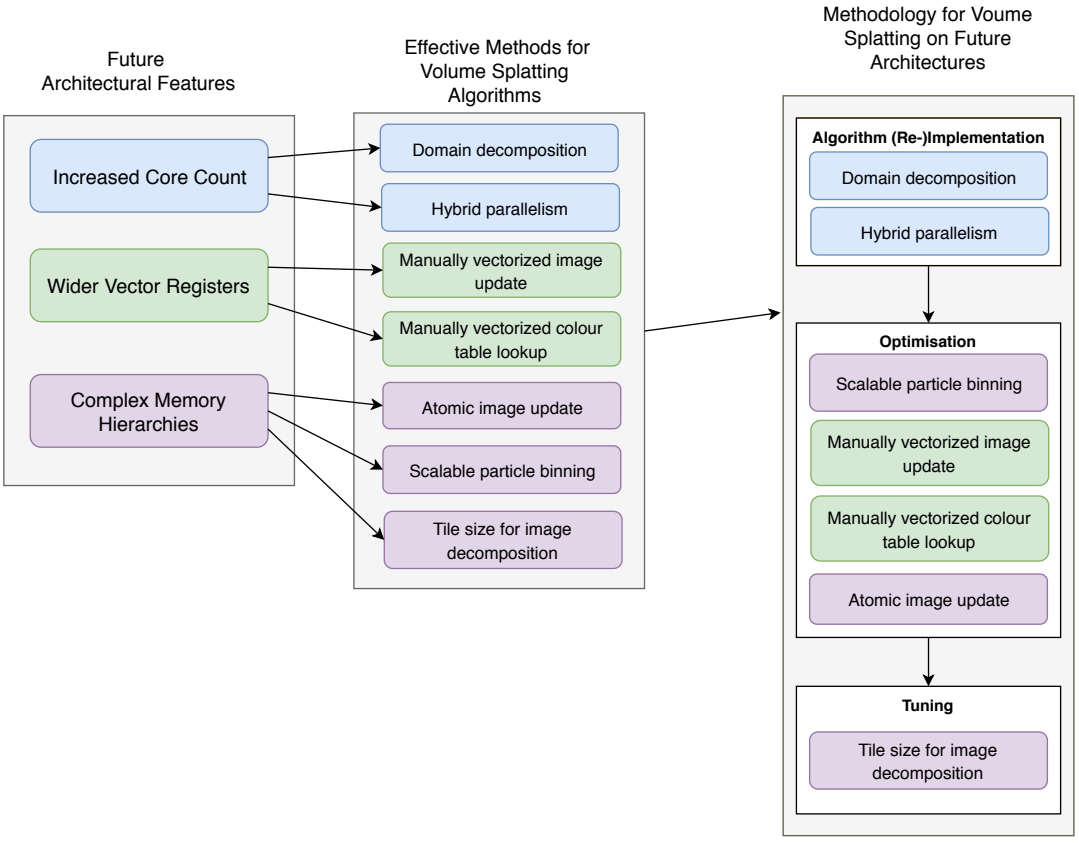


FIGURE 3.36: A methodology for volume splatting on future architectures is presented. First, a relation is defined between the identified future architectural features (figure left) and the methods found effective for Splotch that may be generalised to volume splatting algorithms (figure middle). These methods are then grouped by the stage of the porting and optimisation process they are most relevent to, and ordered to form a methodology (figure right) that may be followed by future implementors.

Overlapped data transfer: Limited memory is a typical constraint on heterogeneous architectures, as such splitting data into chunks to offload to accelerator devices is usually a requirement. Such offload of chunks can benefit from overlapping offload data transfers with execution using device programming model features for asynchronous execution.
Auto vectorization: Help the compiler to automatically vectorize key kernels to take full advantage of the SIMD vector capability of the hardware. This is carried out via detailed analysis of compiler generated vectorization reports, and may require small code structure modifications and inserting hints (typically via pre-processing commands, e.g. *pragma*) to guide the compiler towards appropriate vectorization targets or relax restrictions (e.g. allowing the compiler to ignore potential loop-carried dependencies during vectorization, or allowing the compiler to assume the data is aligned correctly).
Manual vectorization: Where the compiler is unable to infer enough information to automatically vectorize, it is beneficial to manually insert architecture-specific instructions using the wider semantic context known by the programmer. Whilst architecture specific (*intrinsics* are typically carried across a range of architectures per vendor, making this exercise more worthwhile as it is typically applicable to more than a single platform (e.g. Intel Streaming SIMD Extensions (SSE) are pervasive across Intel architectures).
Scalable memory management: The increase in parallelism is likely to have an effect on memory management, especially where allocations to the OS are required. The use of scalable memory allocators designed for lock-free parallel allocation can have a strong impact on performance. Furthermore, using OS features for modifying allocation behaviour may also help, such as modifying the default page size for memory allocation.
Thread count and affinity: As the number of active threads increases, it becomes more important to manage exactly how many threads are running and where they are placed. Performance gains can be realised through experimentation with thread count per core and thread placement techniques such as interleaving, which are typically controllable at run-time and dependent on specific hardware architectures.

Future implementors may follow this methodology, in complement to the application and hardware specific porting process, to aid performance portability on future hardware platforms that follow the expected trend of architectural characteristics presented on the left side of the figure.

3.7.2 Volume Splatting on Future Architectures

Figure 3.36 presents a methodology to be applied when porting or optimising volume splatting algorithms on future architectures, following the expected trend of features presented on the left side of the figure. In the center box of Figure 3.36, the

methods found to be effective for Splotch that specifically concern the volume splatting algorithm are identified, extracted, and again grouped by the appropriate stage of the porting and optimisation process; each of the methods is a specific application of one of the more general methods from the previous methodology, and will be briefly described.

Domain decomposition: In order to integrate an optical model with both emission and absorption, viewing rays must be integrated in correct order relative to the viewer, as described in Chapter 2. In order to achieve performance on large systems, task and data parallelism is required, however this requires a domain decomposition and boundary handling approach for the result to be correct. Section 3.6 presents a method that is applicable for volume splatting algorithms based on point sources, which are more difficult to decompose than grid-based algorithms due to the lack of inherent structure in the data.

Hybrid parallelism: Different types of parallelism can affect performance in different ways. OpenMP based approaches are beneficial for the performance gains through use of shared memory, but can suffer from non uniform memory access penalties, whilst MPI-like approaches are required for large distributed memory systems but may require more communication. Volume splatting algorithms can benefit from modifying the implementation to split datasets across distributed memory systems (i.e. across compute nodes), whilst splitting images in shared memory systems (i.e. within a single multi-socket compute node), and then tuning the ratio of such hybrid parallelism to achieve an optimal configuration as demonstrated in 3.3.3.

Scalable particle binning: A typical approach to parallel volume splatting is to decompose the image into chunks, and sort the objects in bins corresponding to the image chunk they affect. A simple parallel sort is unlikely to scale to many cores; one of the reasons for this can be memory allocation during the sorting. Using a lock-free parallel memory allocator in this case can provide a large benefit, as demonstrated in 3.3.2.2 (I).

Manually vectorized colour table lookup: A standard component of volume splatting algorithms is a colour table lookup during transfer function computation (see Section 2.2.2. This can be difficult for the compiler to vectorize as there is typically a function call required to interpolate colours. In a parallel loop over objects for colour lookup, chunking the loop into the width of the vector register on the hardware platform can allow a manually vectorized interpolation function to be implemented which can perform better than a simply parallelised loop, as shown in Section 3.3.2.2 (II).

Manually vectorized image update: The object order approach of volume splatting algorithms (see Section 2.2.4) requires each volume element to be projected onto the rendered image. As evidenced in Splotch, automatic vectorization is unlikely to apply here as the size of each object is unknown during compilation. One approach that can provide performance, as detailed in Section 3.3.2.2 (II), is the manual vectorization of the pixel writing for rows (or columns) of image pixels during this image

update.

Atomic image update: Following from the previous method, updating the image during parallel volume splatting will result in race conditions if not synchronised properly. As shown in Section 3.4, using hardware provided atomic operations can provide a performance benefit over manual synchronisation methods.

Tile size for image decomposition: Parallel image updating in volume splatting may require the image to be decomposed into tiles, a heuristic based parameter search may be trivially implemented and can be effective to find a default tile size parameter for a specific architecture, which will depend on cache hierarchy and size.

It is expected that implementors of algorithms that follow the volume splatting paradigm for high performance visualisation will be able to utilise this methodology to target specific parts of their algorithm in view of achieving performance on future architectures.

3.8 Discussion

This chapter has introduced a variety of parallel approaches for volume rendering, including optimised usage of accelerators and a distributed memory parallel rendering algorithm. To address the larger picture, effective exploitation of the underlying hardware, whether through distributed memory techniques or architectural optimisation, is a necessity for high performance scientific visualisation due to excessive data size, complex algorithms, and strict performance requirements (see Section 1.2). The life-cycle of hardware in HPC systems can be short; the systems themselves have a typical life-cycle in the region of 3-5 years before the hardware becomes obsolete (Strohmaier et al., 2005). Particularly for experimental hardware such as the Intel Xeon Phi, newer models can quickly supersede older ones (e.g. *Knight's Corner* to *Knights Landing* (Jeffers, Reinders, and Sodani, 2016)), or be discontinued in favour of a new architecture entirely (e.g. the discontinuation of expected future Xeon Phi model *Knight's Mill* (Damkroger, 2017)). As such it is of great importance to focus on hardware trends and common characteristics to future-proof algorithms and implementations where possible whilst staying aware of the unpredictability of one single platforms longevity.

Moreover, the pace at which hardware evolves is considerably faster than that of large application codes, whose lifetimes are '*often measured in decades, rather than years*' (Giles and Reguly, 2014). This parallel, but unsynchronised, progression presents a further challenge for those looking to achieve maximum performance on cutting-edge HPC systems; by the time an application with many millions of lines of code can be ported to a new architecture, it may well be superseded by another, requiring further time and cost expenditure. As such, a serious consideration for porting of codes to a new architecture is the *future-proofing* of these codes for successive architectures. As a pre-cursor to porting large application codes that require huge cost in time and labour, however, the performance and applicability of more

lightweight applications should be evaluated on experimental architectures. In this way, the experiences and knowledge gained in works such as this can serve to build an informed picture of the expected performance of larger applications, the types of workloads that can perform well, and the problems that are likely to be found. In this vein, the methodologies defined in Section 3.7 provide experience-driven recipes that may be followed for larger and more slowly evolving applications to address quickly evolving hardware, focusing on the general features and trends rather than architecture specifics.

In the longer term, there are alternative options that may be worth considering. The concept of *performance portability*, writing one code base that can achieve performance over a variety of architectures, is becoming ever more important. Further investigation is warranted into the viability of novel programming models, or extensions to existing models, to achieve performance portability in a standardised manner, and the use of performance portability metrics such as defined by (Pennycook, Sewall, and Lee, 2016). Field Programmable Gate Arrays (FPGAs) are growing in popularity, as seen by the recent introduction of Intel Stratix 10 FPGAs for HPC, which may increase the interest in OpenCL which is one of only few supported high level languages for FPGA programming. Beyond this, there is increasing interest in exotic computing architectures; for example, approximate, quantum, and neuromorphic computing. For high performance scientific visualisation, as with many other computational fields, there will be a need to evaluate and understand the approaches to exploiting such architectures.

3.9 Summary

This chapter addresses the objective **O.2**: *'Analyse and exploit emerging HPC architectures in the context of high performance visualisation'*.

Section 3.3 details the porting and optimisation process to utilise the Intel Xeon Phi Knights Corner accelerator as an offloading target for visualisation, with comparison to the NVIDA K20X GPU. The work evaluates the effect of various optimisation techniques, including manual and automatic vectorization, hardware event analysis, page size, thread affinity and varying parallel models. The ported code achieves up to 9x performance improvement v.s. the CPU on a per-kernel basis, and results in roughly similar performance for the full algorithm due to overheads such as data transport; the optimised version of the code performs up to 6x better than the initial Xeon Phi implementation. Section 3.4 revisits an existing GPU code to evaluate the applicability of newer architectural developments, namely improved atomic operations on the NVIDIA Tesla architecture. The work simplifies the original algorithm, making use of atomic operations to avoid race conditions when writing to image pixels without the overhead of a complex tiling schema; the adapted algorithm achieves between 1.3x and 3.1x speed-up. Future work is described, taking into account the memory hierarchy of the GPU to show potential for further

speed improvements. This work demonstrates the utility of atomic operations on modern GPUs over manual synchronisation, and illustrates the benefit of adapting algorithms to match architectural evolution of accelerator hardware. In Section 3.5 the work of the Sections 3.3 and 3.4 are quantitatively compared with commentary. Section 3.6 presents a parallel algorithm for particle splatting, as a proposed extension to the work of Chapter 2, to address the distributed memory hierarchy typical of high performance computing systems. Section 3.7 presents two methodologies based on the experiences of the previous sections, aimed at identifying, extracting, and grouping general and splatting-specific methods for future implementors. The work presented in this chapter (specifically, Sections 3.3 and 3.4) is also presented in (Dykes, Gheller, Rivi, et al., 2017) and (Dykes, Gheller, Krokos, et al., 2015).

In view of the question posed: **Q.2**: *'How can high performance visualisation cope with modern heterogeneous high performance computing systems?'*, this chapter has addressed the characteristics of hardware present in current HPC systems that are expected to prevail in future heterogeneous architectures to present a route forward for general, and high performance visualisation specific, algorithms.

Chapter 4

Remote Interactivity for HPC

This chapter aims to answer the question **Q.3**: *'How can modern astronomical tools exploit remote and interactive high performance visualisation on the web?'*, by addressing the corresponding objective **O.3**: *'Explore a general approach to addressing web-based interactive high performance visualisation'*. This chapter begins with a brief introduction to interactive HPC and the web (Section 4.1), expanding on the topics introduced in Section 1.3.3. The current state of the art for interactive and web-based high performance visualisation software is then reviewed in the context of a novel classification scheme for remote applications. A general approach to facilitate web-based remote interaction with HPC applications is developed (Section 4.2). This approach is then used to build a web-based, interactive, remote visualisation tool in Section 4.3, providing the foundations for the work of Chapter 5.

4.1 Interactive HPC and the Web

In the current Internet era, much of the information we need throughout our working day is available on the web. Over the past few decades communications infrastructure, user hardware, web browsers, and associated tools and libraries have evolved to such an extent that many user applications that once would have been installed locally can now be run entirely remotely. This new, remote, web-based, paradigm means that an Internet user can now do much more than type emails and view static pages in their browser, as was the case in the early days of the Internet. They can now type an entire Ph.D. thesis through online word processors, edit photos with image manipulation tools, compete in online 3D games via real-time 3D rendering and streaming, design engineering solutions with web CAD applications, and more, all from the comfort of a laptop web browser. This paradigm has led to a user expectation of powerful web services, exploiting *cloud computing* (Rimal, Choi, and Lumb, 2009; Patidar, Rane, and Jain, 2012), that are more accessible, portable, and simpler to use, than traditional native applications.

Web browsers typically do not require expensive high powered hardware, and are pre-installed on most personal computing devices. Users can access services built and hosted on the other side of the world to communicate, run applications, and store and retrieve data in real-time, without prior installation or configuration

on their local machine. With the rise of low cost, on-demand, cloud-based web infrastructure, for example Amazon Web Services (AWS) and Microsoft Azure, and fast internet streaming speeds, it is becoming commonplace for fully featured Software As A Service (SaaS) applications to be available in-browser (e.g. (Miller, 2009)).

Conversely, HPC applications are traditionally executed remotely by exploiting computing clusters accessible by terminal or remote desktop protocol (e.g. Virtual Network Computing (VNC) client or X Forwarding protocol). User applications are run via job requests submitted to a resource management system, which are queued and executed when the required resources become available. This approach is effective for many traditional HPC applications (e.g. modeling and simulation); however, due to this paradigm of web and cloud computing, the modern day research scientist (a typical HPC user) has access to a much broader array of tools that may benefit from, or even require, high performance computing, from interactive computing with Python-based tools to workflow software for coupling and/or chaining multiple large scale parallel applications.

Notably, there are a variety of HPC applications that necessitate a Human-In-The-Loop (HITL) (Nunes, Zhang, and Silva, 2015) with real-time user interaction; for example, visualisation software, monitoring and inspection utilities, debugging tools, and computational steering software. This type of application requires some form of user interface, which is typically achieved through a client-server remote software approach where a local user application communicates with a remote HPC application; for example, a local graphical debugging client connected to a debug server running on a HPC system. However, these approaches are often tailored to a specific application and require a significant amount of setup (detailed further in Section 4.2.2). This class of application could benefit from the accessibility, portability, and simplicity of cloud-like web-based software.

High performance visualisation is typically performed remotely, through a visualisation server exploiting HPC resources to process large datasets and a client running on a user machine. It is essential such remote tools can integrate with the modern research environment, as introduced in Section 1.3.3, such that a scientist can visualise their data as part of their scientific workflow. However, there is a lack of interoperability between web environments and high performance computing environments, which poses difficulties for the development of remote tools that can integrate with the modern research environment. This chapter first addresses this lack of interoperability between web and high performance computing in a general case, through a library for remote interaction with HPC applications detailed in Section 4.2. This library is then exploited to build a remote, web based, interactive scientific visualisation tool based on Splotch in Section 4.3.

4.2 Remote Interactivity for HPC

4.2.1 Interoperability for Web and HPC

As discussed in Section 1.3.3, there are examples of an ongoing convergence of web and traditional HPC environments. This includes creating HPC-like environments and running HPC applications on cloud-computing infrastructures (Canon et al., 2010; Gupta and Milojicic, 2011; Church, Goscinski, and Lefèvre, 2015), building cloud-like infrastructures on HPC systems (Mauch, Kunze, and Hillenbrand, 2013), web-based applications exploiting HPC resources such as JupyterHub (Section 4.2.3), HPC applications exposed to users through the web via workflow management tools (Goecks et al., 2010; Brown et al., 2015) and *Science Gateways* (Gesing et al., 2015), and even public web platforms supported by HPC infrastructures such as the Theoretical Astrophysical Observatory (Bernyk et al., 2016) and similar efforts in scientific communities. Cloud-like environments are further enabled through container solutions for HPC such as Shifter (M. and S., 2015) and Singularity (Kurtzer, Sochat, and Bauer, 2017), paired with containerized solutions for HPC applications such as NVIDIA's GPUCloud[1].

There are existing efforts to provide cloud-like services for HPC, from creating HPC-like environments or running traditional HPC applications on cloud-computing infrastructures (Canon et al., 2010; Gupta and Milojicic, 2011; Church, Goscinski, and Lefèvre, 2015), to building cloud-like infrastructures on HPC systems (Mauch, Kunze, and Hillenbrand, 2013). This extends to HPC applications exposed to users through the web via workflow management tools (Goecks et al., 2010; Brown et al., 2015) and Science Gateways (Gesing et al., 2015), and public web platforms supported by HPC infrastructures such as the Theoretical Astrophysical Observatory (TAO) (Bernyk et al., 2016) and similar efforts in other scientific communities. However, for developers to build HPC applications requiring real-time interaction, interoperability between web and HPC technologies is a necessity, particularly in terms of auxiliary software libraries. This is challenging especially due to the significantly disparate environments, both software and hardware, of web and HPC. A domain scientist developing high performance research software is unlikely to also be an expert in web development or even familiar with the languages and tools common in the field, and vice versa. This difficulty is compounded when there is a requirement for real-time interaction, necessitating high performance implementations on both sides. There are only few examples of HPC applications and web applications interacting in real-time (see Section 4.2.3); whilst this is partially due to the infancy of a HPC-Web convergence, the problem is exacerbated by a lack of general purpose tools to facilitate interoperability.

This challenge of interoperability is already being addressed in the HPC community, encouraged by projects such as NEWT (Cholia, Skinner, and Boverhof, 2010) for web-based interaction with resource management systems, environments such

[1] https://www.nvidia.com/en-us/gpu-cloud/

as EnginFrame[2] and Bridges (Nystrom et al., 2015) that are designed to support web portals and non-traditional HPC workloads, and containerized solutions packaging HPC software for general portable usage such as those available on NVIDIA's GPU-Cloud[3]. It is clear there is an emerging paradigm shift in HPC from the traditional command line batch scheduled job execution to a more accessible and user-friendly experience motivated by web, cloud, and interactive technologies. However, there is still a long distance to go, particularly for applications requiring real-time user interaction.

To help bridge the gap between HPC and Web applications and address the lack of general purpose interoperability tools, the following subsections present WSRTI: a WebSocket (Fette and Melnikov, 2011) based framework for fast data streaming and simple user interaction with active HPC workloads. The core principle of this framework is to allow HPC specialists and research scientists to quickly and easily create web interfaces to monitor and interact with active HPC applications in real-time. The framework includes a lightweight mechanism for a headless HPC application to expose a Remote Procedure Call (RPC) interface automatically linked to a web based graphical user interface. This is complimented by data streaming support to allow the user to transport data to and from the application independently of the RPC mechanism.

The remainder of Section 4.2 is structured as follows: Section 4.2.2 presents a novel classification scheme to identify and differentiate remote applications. With reference to this scheme related work is discussed in Section 4.2.3, before presenting the WSRTI framework in Section 4.2.4. This is followed by a practical explanation of the requirements for a HPC application to utilize WSRTI in Section 4.2.5, with a brief discussion of performance in Section 4.2.6.

4.2.2 A Novel Classification Scheme for Remote Applications

The inherently remote nature of working on a HPC system can discourage use of interactive applications. HPC systems are traditionally hosted in dedicated computing centers and on university campuses, and access typically requires a local workstation and terminal connection via Secure Shell (SSH) protocol. Once connected, the user can submit jobs through a resource management system such as SLURM (Yoo, Jette, and Grondona, 2003) or the Portable Batch System (PBS)[4] to request computing resources in either a *batch* or *interactive* manner. *Batch* submission inserts the job into a queue and schedules it to run when resources are available to be allocated, which allows efficient job management by the scheduler and maximum resource utilization. However, the job must be defined ahead of time and often failure will yield the allocation and require a new job to be created. *Interactive* submission indicates the resources are required immediately, providing the user with a shell where they

[2] https://www.nice-software.com/products/enginframe
[3] https://www.nvidia.com/en-us/gpu-cloud/
[4] e.g. http://www.pbspro.org/

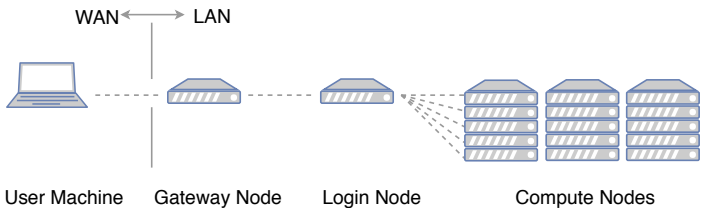


FIGURE 4.1: The typical layers of indirection between a user machine and the compute nodes of a HPC system.

can interactively launch applications for the duration of the allocation. While it is standard practice to support an interactive queue for applications that require it, interactive applications typically have a broader set of requirements, e.g. supporting user interaction.

An important characteristic of an interactive HPC application is the means by which user interaction is supported. This can be complicated by one or more layers of indirection that typically exist between the user and the compute nodes on which their software executes, illustrated by Figure 4.1. This indirection is necessary for security of the system, however can introduce difficulties for the end user who is routinely also outside of a gateway firewall. Furthermore, compute nodes typically have a stripped down operating system with only high performance components, as such may not include software for traditional desktop environments (e.g. an X11 server). Interaction with software running on the compute nodes must somehow address these layers of indirection between the user and the application. The most common approach is for the remote application to act as a server, and a local application on the user's machine to act as a client, coupled via a communication schema and forwarding mechanism such as SSH tunneling. However, there are a variety of different approaches for this client-server scenario, from remote-desktop software to bespoke application frameworks.

First the approaches for remote interaction with high performance applications are classified, in terms of *direct* vs. *indirect* approaches, and *web* vs. *native* approaches. This classification scheme is outlined in Figure 4.2, in the context of a typical HPC setup. In this diagram, the high performance application is labeled *App Server*, and is connected in a variety of manners to an interface on the user machine. Indirection of an approach refers to the use of auxiliary software (*Aux*) to support the remote access. Applications are connected via an *Application Layer Protocol* (ALP) (Zimmermann, 1980), and each *hop* refers to a jump between network layers that may need facilitating (e.g. by port forwarding).

Direct Remote Native (DRN): This classification refers to native applications that implement the full end to end client-server model, i.e. the application interface is installed on a users local machine, and the application server runs on the compute nodes of a HPC system. The connection is typically a custom messaging schema sent over a common application layer protocol (e.g. ZeroMQ[5], TCP sockets), and

[5] http://zeromq.org/

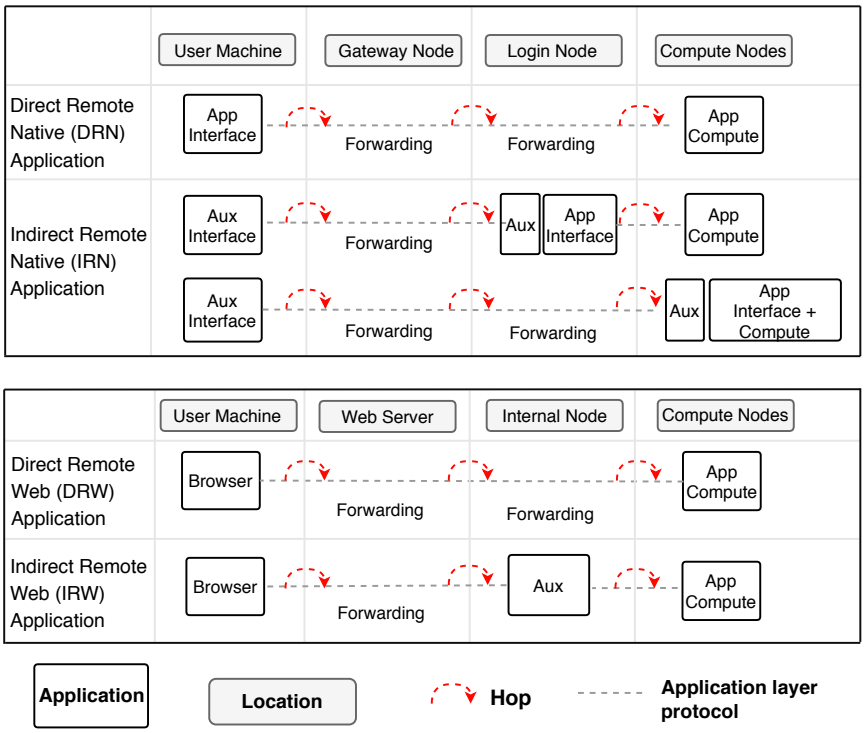


FIGURE 4.2: A classification scheme for remote applications in the context of HPC systems. *Top*: Native applications (DRN, IRN), *Bottom:* Web applications (DRW, IRW)

each hop is often enabled via SSH tunneling. This approach can perform well due to developer optimised messaging schemas tailored for the specific application, however relies on availability and installation of a client for the specific user machine. The increased performance is more effective for interactive applications, particularly those requiring real-time interaction. Examples of this class of application are high performance visualisation software tools ParaView (Ahrens, Geveci, and Law, 2005) and VisIt (Childs et al., 2011), and remote debugging software such as NVidia NSight Eclipse edition (NVIDIA Corporation, 2018a).

Indirect Remote Native (IRN): In this case native applications implement part of the client-server model, however may rely on auxiliary software to extend fully to the user machine, or do not directly connect to the remote application. There are various indirect approaches, and the location of the interface or auxiliary software does not always match exactly the two example scenarios shown in Figure 4.2. A typical example of the first scenario is a client application which runs on the login node of the HPC system in order to interact with the resource management system; the client is then accessed locally via a remote desktop approach such as TurboVNC or X Forwarding. The second scenario requires the interface and auxiliary software to run directly on the compute nodes; this can be more difficult to achieve due to the lack of common desktop environment software on compute nodes. Both approaches can provide more portability when using auxiliary software. For example, a linux client can run on the HPC system whilst the user connects through remote desktop software on a Microsoft Windows-based PC. However, performance can degrade as

auxiliary software is typically not able to optimise data transfers as effectively as custom approaches in a Direct Remote Application, and may use slow transfer protocols such as reliance on a shared filesystem. Examples of this type of application are remote profiling tools such as Intel VTune Amplifier (Intel Corporation, 2018) or the NVidia Visual Profiler (NVIDIA Corporation, 2018c). This approach is also commonly used to allow installing a variety of user software, which may in fact be direct remote applications, on a HPC system, such that the users of the system only need to install a remote desktop solution on their own machine.

Direct Remote Web (DRW): A web application running in the users browser is directly connected to the application server running on compute nodes. The direct approach requires exposing a public web server that also has internal access to the compute nodes, and directly connecting the user web application to the server via a web-capable ALP such as WebSockets (the direct connection may include intervening router software). This approach is most flexible and convenient for the user, however introduces significant security concerns if the web-server is to be made public. A simple work-around for an internal application is to run the web server on an internal node, and use SSH Tunneling to forward ports from the user machine to the internal node, thus securing the user connection via SSH. Some examples of these approaches are the ParaView Web Interface and JupyterHub (Jupyter Hub Development Team, 2018), detailed further in Section 4.2.3.

Indirect Remote Web (IRW): A web application running in the users browser is connected via a web server to auxiliary software (beyond simple routing software), which in turn is connected to the application server. This approach allows for the auxiliary software to provide a layer of security between the public web server and the application, however can introduce difficulties for interactivity. For example, common techniques are to use a shared filesystem or database polling as an intermediary step to pass requests from the user to the server, which is sufficient for data access but introduces additional latency and constraints not amenable to real-time interaction with the application. This approach is typical for public facing web portals supported by HPC resources, for example TAO (Bernyk et al., 2016) and the Cosmological Web Portal (Ragagnin et al., 2016) (for more on these, see Chapter 5).

This four-part classification scheme effectively describes the ways in which applications are run remotely on HPC systems. Some may support more than one mode of execution, and so can fit into multiple categories of this scheme; for instance extensive visualisation software ParaView can be used with various types of interface that can be classified in each of these categories.

4.2.3 Related Work

With reference to the classification scheme of Section 4.2.2, there are many remote applications that can be placed into one or more of these categories. The scope of this work is limited to those that fit into the Direct or Indirect Remote Web categories

and can support real-time interactivity. The related tools that have been identified as fitting into the DRW or IRW classifications are listed below.

ParaView (Ahrens, Geveci, and Law, 2005) is a large scale parallel visualisation software, designed for effective exploitation of HPC systems. A web enabled version ParaViewWeb[6], can act as a Direct Remote Web Application by allowing the user to remotely connect via web browser to a ParaView server running on a HPC system. The connection is enabled via custom library *wslink*[7] that connects JavaScript web clients to a Python web server through ALP WebSockets. Furthermore, the in-situ library Paraview Catalyst (Ayachit et al., 2015) allows users to instrument their application for in-situ analysis, visualisation, and computational steering purposes.

The Cactus computational framework (Goodale et al., 2003) supports a web browser interface for in-situ visualisation and steering tasks. The user can instrument existing HPC applications with the Cactus API, and perform interactive steering tasks and view visualisation outputs through a web browser. The standard implementation utilises a HTTPD [8] web server, and forwards ports to the user for remote access.

The Jupyter Notebook[9] is a web application allowing users to interactively write and execute code, and transform, analyse, and visualise data using a variety of languages. The Notebook can be set up manually on a HPC system, exposing a web interface via the built in Notebook python web server that can be accessed by browser from a user machine via port forwarding. Furthermore, it is possible to deploy JupyterHub as a multi-user hub to access over HTTP, an approach also viable on HPC systems (Milligan, 2017).

ViSUS (Pascucci et al., 2012) is a portable visualisation framework designed to run on many platforms, from mobile to cluster systems. A ViSUS viewer can be compiled as a browser plugin, whilst a ViSUS server can be implemented as an Apache web server plugin running on a cluster machine, allowing the simple integration of a high performance ViSUS viewer and server for web applications. The interface is based on a stateless HTTP scheme, and supports multiple clients as an IRW approach.

The WebVis framework (Zhou, Weiss, et al., 2013) is a multi-user, client-server, visualisation system with a web-based client that can interact with a cluster-based visualisation server. The client is connected to the server via a back-end service built using the Google Web Toolkit and a Java web server, which communicate with an institutional web service that forwards events and images to and from the internal render cluster (i.e. an IRW approach). Client GUI interactions are forwarded to the server, and images returned, through an EventBus using the *HTTP server push* paradigm.

[6] https://kitware.github.io/paraviewweb
[7] https://github.com/kitware/wslink
[8] https://httpd.apache.org/
[9] http://jupyter.org/

Tapestry (Raji, Hota, and Huang, 2017) is a recent work focusing on seamless embedding of interactive volume rendering in web pages, by loosely coupling a Docker container based rendering service with a Tapestry object embedded in a HTML <*img*>. Requests are passes from the browser to the rendering service via HTML *GET*, passed to a web server and rendering service running inside a docker container. Many instances of such containers can run on a computing cluster, exploiting task-based parallel volume rendering. The main focuses of Tapestry are *scale of audience*, to support interactive and volume rendered web embedded content for many users, and effective embedding in the standard web Document Object Model.

The commercial remote desktop software FastX[10] enables users to use existing desktop interfaces via the web with a WebAssembly module for a uniquely efficient high performance remote desktop service in the browser. This option is an effective solution for enabling access via the web for applications with existing interfaces, however does require a FastX license. A similar approach can be taken with other remote desktop protocols, such as VNC through e.g. TightVNC[11]. Furthermore, compute nodes of HPC systems commonly have a stripped down version of Linux that does not support GUI applications, so in most cases the application must already support running from a login node. This software can enable Direct and Indirect Remote Applications to be used as Indirect Web Applications.

Remote frameworks that support web such as FastX are seeing some success, for example the web visualisation portal at the Texas Advanced Computing Centre supports web-based usage of Paraview through a web-based VNC session, however such approaches do require the user to already have a direct/indirect remote application. Other solutions are built upon bespoke frameworks which are effective for a single application but less useful for the general case of a HPC application requiring remote interaction or monitoring. The development of WSRTI, described in the next section, is intended to address the lack of general purpose tools for interoperability, in order to support the development of more applications using the DRW and IRW approach for remote interactivity.

4.2.4 A Framework for Connecting Real Time HPC Applications to the Web

The WSRTI library is built to facilitate interoperability of web and HPC technologies. The aim is to reduce as much as possible the burden on a research scientist to understand web technologies, and allow them to link their application to a web interface automatically from application-side specifications. The key features supported are data streaming, RPC and event forwarding, and dynamic interface generation. The following subsections discuss the technical details of the library and illustrate how each of the key features is supported.

[10] https://www.starnet.com/fastx/
[11] https://www.tightvnc.com/

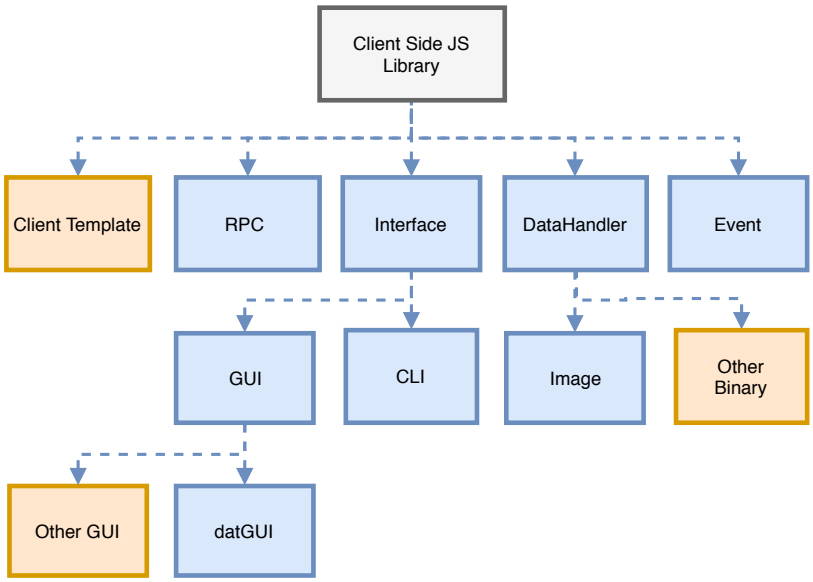


FIGURE 4.3: Structure of the client-side JavaScript library.

4.2.4.1 Overview

WSRTI is conceptually split into two components, a client side web library and a set of application-side C++ utilities to assist HPC developers in exposing RPC functionality and application data. The two components are connected via a WebSocket communication scheme discussed in Sections 4.2.4.2 and 4.2.4.3.

The client-side library is written entirely in JavaScript, exploiting the most recent features of the ECMAScript 6 standard. Figure 4.3 illustrates the client-side structure, consisting of Data Handler, Event, RPC, and Interface modules (further described in the following sections). These modules are integrated into an included client template, a web client capable of receiving and displaying images with a console input and debug log.

Figure 4.4 illustrates the collection of server-side utilities, consisting of modules for binary serialization, generation of JSON interface objects, RPC, and asynchronous queueing, along with convenience wrappers for image compression and WebSocket servers. These modules can be used by the HPC application developer to export data and interface descriptions, while sending and receiving RPCs and events. All modules of the library are documented with Doxygen[12], and contain unit tests that also act as example usage.

4.2.4.2 Communication

In order to facilitate fast and interactive web communication, WSRTI exploits the WebSocket protocol. The full-duplex nature of a WebSocket connection is ideal for interactivity; messages can be sent and responses received without polling, allowing streaming services to be built in a simple and efficient manner. Figure 4.5 illustrates

[12]http://www.stack.nl/ dimitri/doxygen/

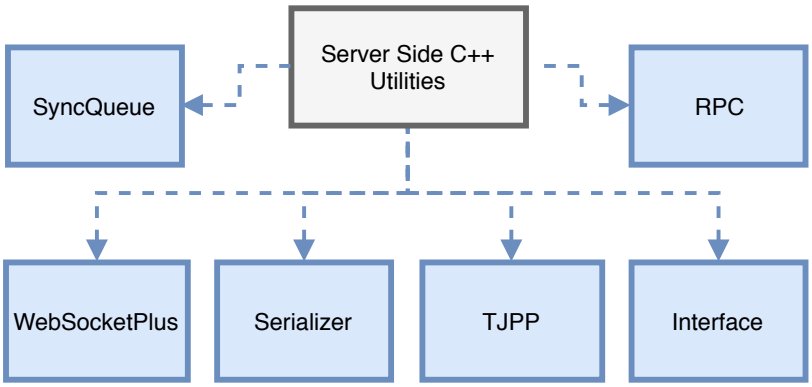


FIGURE 4.4: The collection of C++ utilities for HPC applications.

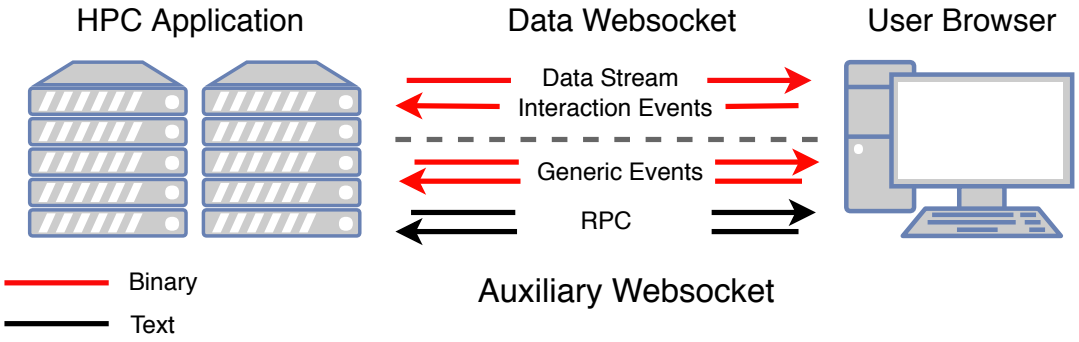


FIGURE 4.5: The data flow between a HPC application and WSRTI. Data is split across two sockets, one for dedicated binary streaming and a second for binary event and text RPC messages.

the data flow in WSRTI. Two sockets are created to connect to the application, a dedicated Data WebSocket for streaming application data and user interaction events, and an Auxiliary WebSocket for RPC commands and binary event transfers.

The Data Websocket sends and receives binary messages. Generic data can be sent from the server to the client, which is received by a specific client *Data Handler* (Figure 4.3). User interaction events are sent by the client Event module back to the HPC application (Section 4.2.4.3). The Auxiliary WebSocket handles a mixture of text and binary messages. Generic binary events and text based RPC messages can be forwarded back and forth between HPC application and user browser.

Two dedicated sockets allows WSRTI to make assumptions about the type of message received and optimize client-side de-serialization. This approach is used for data streaming from application to client, which may potentially trigger messages in high frequency and volume. For example during image streaming the client data handler can choose to interpret all messages received as JPEG compressed images and directly update the display at high frame rate. This avoids the need to de-serialize and check a message identifier which can have a noticeable impact on performance in the JavaScript data handling module.

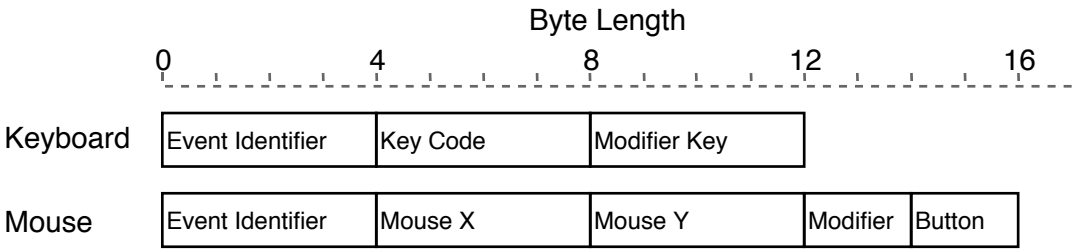


FIGURE 4.6: The byte structure of two example binary events: Keyboard and Mouse.

4.2.4.3 Data and Event Streaming

Client and application side utilities enable the user to package and asynchronously stream data back and forth as a generic binary data stream as well as formatted binary messages between the active HPC application and web client. The user can exploit the dedicated *Data Stream* to forward application data to the client, and create generic binary events to signal the application, or the client, that a particular software event has occurred.

The client data handler consists of an *active* data receiver that monitors the Data Stream. Data receivers are simply implemented, and assume the message they receive is of an expected format. The client and application should both agree on the type of data expected on the Data Stream at any one time (this can be set, for example, by a binary event). The data handler receives binary messages on the Data Stream in the form of a JavaScript *Blob*, and forwards to the current active data processing module, which is by default an image processor.

The default image processor is implemented as part of the template web client, and utilizes a double buffered approach common in graphics rendering. As a message is recieved, the handler generates an object URL which is stored in a secondary data buffer, the front and back buffers are swapped, and the displayed image is updated. There is also a default generic data processor, which can ping-pong binary data packets.

The client event handler receives binary and character messages (distinguishable in the WebSockets layer). Binary events are stored as JavaScript *TypedArrays* with a preceding four byte integer identifier, using a schema agreed upon by both client and application. This identifier is inspected and the event is forwarded to a specific event handler, for example this may be a non-standard data message (e.g. a one-time downloadable file). Character messages are inspected for valid JSON-RPC formatting and forwarded to the RPC module. Internally, WSRTI can forward binary interaction events to the server from the client browser window, such as keyboard, mouse and window activity. These are received on the Auxiliary Websocket on the server.

The application-side *serializer* utility is a C++ header-only library to assist with serializing C++ objects and structures to binary messages, for example to create

```
// JSON-RPC function request
{
   "jsonrpc": "2.0",
    "method": "load_data",
    "params": { "filename": "/path/to/file.ext" },
    "id": 1
}
// JSON-RPC function response
{
   "jsonrpc": "2.0",
    "result": 0,
    "id": 1
}
```

FIGURE 4.7: A JSON-RPC formatted request and response. The request contains a function name and series of parameters, with an additional id used to match responses to requests if a response is required.

events or serialize application data. The *tjpp* convenience library assists with compressing images to JPEG via lib-jpeg-turbo[13]. These can then be passed to a WebSockets server (*WebSocketPlus* utility) via a synchronized queue (*SyncQueue* utility) to be asynchronously sent to the client.

4.2.4.4 Bi-directional RPC

The client RPC module and the application can exchange RPC requests in both directions. However, it is a burden to have to implement application specific RPC handling on both client and application, and ideally the client needn't know in advance a list of specific RPC functions and arguments. In order to avoid this, the application developer can specify a function name and list of arguments as part of the interface generation process (see Section 4.2.4.5 for more), binding an interface element such as a button or input field on the client to a procedure call on the server at application run-time. This also allows the application RPC interface to be extended or updated without changing the client code.

Internally, the RPC module can send and receive JSON-RPC[14] formatted requests. Bi-directional RPC allows the server to also trigger client-side operations if supported, such as handling a dynamic interface descriptor or accepting a file download.

The JSON-RPC format is a simple specification that covers the necessary feature set for RPC in this context. JSON (JavaScript Object Notation) is essentially a subset of JavaScript, and natively supported by the language. For a JavaScript based client library, JSON-RPC is a natural format for use. Figure 4.7 demonstrates the structure of a JSON-RPC formatted request and response. The *id* parameter is used to match requests and responses, required for *request-reply* communication, while a null or non-existent *id* parameter indicates a *notification* that does not necessitate response.

[13] https://libjpeg-turbo.org/
[14] http://www.jsonrpc.org/specification

This bi-directional model of RPC allows more advanced clients to be implemented if preferred, for example facilitating a peer-to-peer model where a client can trigger an action that will be forwarded to multiple other clients. This feature would support an advanced implementor in creating a collaborative environment through indirect messaging between clients.

4.2.4.5 Dynamic Interface Generation

The library follows an application-centric design, meaning that the majority of the client functionality is defined by the application at runtime through dynamic interface generation and RPC linkage. This allows the developer to focus on their HPC application and the data they want to expose, rather than on building a web GUI. The HPC application at runtime can define and modify the look and feel of the web client user interface, for example adding or removing buttons and sliders and specifying RPC calls and arguments that should be linked to client actions.

The interface generation mechanism depends on the construction of an *interface descriptor*, typically on the application side, which is forwarded to the client on connection. This can then be updated via client-targeted or broadcast RPC messages from the application, or upon request by the client. The descriptor is a hierarchical structure, reflecting the visual hierarchy of a user interface, that describes a series of buttons, sliders, input fields and display fields.

Figure 4.9 demonstrates the application-side process to create an interface descriptor. Groups of menu elements are of type *group*, while elements have types such as *Button* or *TextInput* which are used by the client to generate web interface elements. The *args* object is used to specify arguments to the RPC call, which are links to data objects contained in the interface descriptor and can be generated via menu interactions or manually. The JSON descriptor is generated using a language specific JSON library, such as RapidJSON[15] for C++, an example of the generated interface descriptor is illustrated in Figure 4.8. A C++ *Interface* header included with the framework provides utility functions to generate JSON menu elements such as text input boxes, numerical sliders, and buttons, whilst an *RPC* header supports creation of RPC messages.

The client interface module (Figure 4.3) is split in two, a CLI component and a GUI component. In the template example, the CLI component is linked to an input box on the web page, and parses the input string for internal commands (such as a request to print the help message to the log) distinguished by a preceding '.' character, or commands to the server which are forwarded as RPC calls. This is helpful for very simple text interfaces and debugging.

The GUI component generates an interface based on the interface descriptor. By default, the interface is generated via lightweight graphical interface library datGUI[16], however this can be replaced by a different interface library (e.g. React,

[15] http://rapidjson.org/
[16] https://github.com/dataarts/dat.gui

```
// Top level menu group
{ "User Settings":
  { "type":"Group",
    "contents":
    // Menu elements: leaf nodes of interface descriptor
    // A blank text input box labelled 'Input File'
    { "Input File":
      {  "type":"TextInput",
         "meta": { "value":"" }
      },
      // Button linked to RPC 'cmd_load'
      "Load":
      { "type":"Button",
        "meta":
        { "rpc":"cmd_load",
          "args":
          { "Arg0":"User Settings/Input File" }
} } } } }
```

FIGURE 4.8: A simple example of a hierarchical JSON formatted interface descriptor. This descriptor can be built at runtime via JSON auxiliary library as demonstrated in Figure 4.9

.

jQuery) by overriding a series of interface generation functions. This allows users with web-experience to exploit more extensive interface libraries to add widgets and other advanced interface elements.

Finally, Whilst the client typically parses the interface descriptor from the application and dynamically generates a UI, it is also possible to manually construct this on the client-side if preferred.

4.2.5 Exposing a Remote Application for Interaction via WSRTI

In order to interact with an active remote application, there is a set of requirements the application should satisfy. As interaction typically a necessary part of computational steering, the following requirements take inspiration from those for steering libraries (Brooke et al., 2003). However, WSRTI is intended to apply more generally to remote interaction with applications, as opposed to steering of numerical simulations, as such the requirements are generalised to the minimal requirements for user interaction. At least one of these requirements must be met in order to enable remote interaction:

1. Expose a representation of application state
2. Expose a representation of application data
3. Expose an RPC interface
4. Accept input from standard HIDs

Supporting **(1)** is a minimal requirement allowing a web interface to display the current application state. This could be, for example, whether the application is

```
// JSON Value objects, a top level descriptor, a group representing a subfolder and
    its contents.
Value dsc(kObjectType);
Value user_grp(kObjectType);
Value user_content(kObjectType);
std::string current_input_file = "";

// Create the contents of a 'User Settings' menu group with a text input box for an
    input file name, and a 'load' button
user_content.AddMember("Input File", ui_text_input(current_input_file));

// 'args' indicates the argument to the 'Load' function will be 1 string: the
    contents of the 'Input File' text input box
std::vector<std::string> args = { "User Settings/Input File" };
user_content.AddMember("Load", ui_button("cmd_load", args));

// Add the contents to the user settings group
user_grp.AddMember("type", "Group");
user_grp.AddMember( "contents", user_content);

 // Add the 'User Settings' group to the top level descriptor
dsc.AddMember("User Settings", user_grp);
```

FIGURE 4.9: Creating an interface descriptor with one sub-menu called "User Settings", containing a text input box and a button to load an input file by name. The generated JSON descriptor is shown in Figure 4.8. The syntax shows use of the rapidJSON library with custom memory allocator arguments removed for simplicity.

.

still running and some indicator of algorithmic progress, e.g. the current time in a computational simulation. **(2)** enables an interface to display a representation of the applications working data. This could be a subset of simulation data for analysis, or pre-generated analyses such streaming 3D visualisation or graph plots. **(3)** enables linking web interactions to an RPC interface to trigger application mechanisms such as modifying variables or changing state. **(4)** relates to standard input from Human Interface Devices (HIDs), commonly used in interactive applications to link keys and mouse interactions to actions within the application. A typical use of **(4)** is control of a virtual camera during visualisation or stepping through program execution in a debugging tool.

In order to support one or more of these requirements, the application must contain a control loop or checkpoint, in which point actions **(1)** and/or **(2)** can be performed, or input from the remote client can be accepted and handled. A typical case of control loop is iterating over the time domain during a simulation (or time stepping), in which case WSRTI could be used for computational steering.

These requirements may be satisfied by *instrumenting* user code with a set of additional functions. Figure 4.10 demonstrates the minimum necessities to add interactivity to a generic application for WSRTI. *setup()* is responsible for initialising the WebSocket servers and providing callbacks for event handling, along with

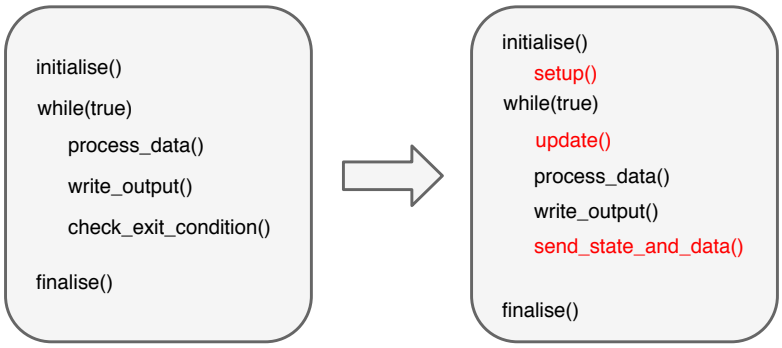


FIGURE 4.10: The generic case for instrumenting a high performance application for remote observation or interaction.

constructing the JSON interface descriptor which is sent to the remote client. *update()* receives events and RPC requests from the client and processes them accordingly, potentially modifying the user code parameters based on the events received. *send_state_and_data* outputs application state and data for the client to process, either in the form of interface updates or streaming data.

4.2.6 Performance

HPC systems are often accessed from remote locations, from elsewhere in the same building to continents on the other side of the world. For this reason, this section presents a number of tests spanning a wide physical area to demonstrate the extent of support for event and data streaming through WebSockets for WSRTI.

For many network-based tools, performance is significantly dependent on the performance of the network. In this case, the most significant factor is the Internet connection between the user and the HPC system, which can vary greatly. As such, real-world performance for WSRTI is dependent on the user and their environment as well as the typical load on the network at any one time. This section describes the performance for one such environment, which represents a typical user scenario. A user laptop is set up as a client in a U.K. University laboratory, whilst a server application is executed on a HPC system at a remote computing facility.

The test system is Swan, a Cray XC50 located at the Cray computing facility in Wisconsin, USA. Each utilized node consists of 2 Broadwell 22-core Xeon CPUs clocked at 2.2Ghz. The web client runs on a Macbook Pro (early 2013 model) with 2.7 GHz Intel Core i7, and Mozilla Firefox 61.0.1, at the University of Portsmouth in the UK.

The test environment on the HPC system is a C++ application (included with the library) that generates data buffers of varying size and streams them over WebSockets using the *SyncQueue* and *WebsocketPlus* utilities. It has the capacity to send without expecting reply, mimicking a data stream, or wait for replies and measure send and receive latencies mimicking events or RPC calls. On the user laptop the template web client containing an additional generic client DataHandler (see 4.2.4.3), which

can act as a binary data stream receiver or a ping-pong type application that will simply return each received message.

The test system network is organised similarly to Figure 4.1, without the gateway node. A port is forwarded for each WebSocket via SSH tunnel, e.g. '`ssh -fL port1:nodeID:port1 -L port2:nodeID:port2 -N user@login-node.domain.`', and the client web-page is hosted locally for the user. For a series of tests to measure latency and bandwidth, packets of varying size are streamed to the client.

Figure 4.11 shows a series of bandwidth results for the described test setup. In this environment a maximum sustained bandwidth of ˜9 MB/s is reached for data streaming, which is reached at packets of 2MB, with ideal bandwidth for packet ranges from 128KB to 2MB. As previously mentioned, test results are dependent on the current network load, and so may vary with time.

Figure 4.12 shows data latency, for packets under 8KB a latency of under 100ms is seen, which is acceptable for real-time interaction in most applications, although may be considered high for very high performance interactions such as those necessary for competitive online gaming. For those types of application, developers endeavour to use servers physically closer to the user than shown in this example. For packets up to 500KB latency remains in the 1-200ms range, steadily rising as packets increase in size beyond this.

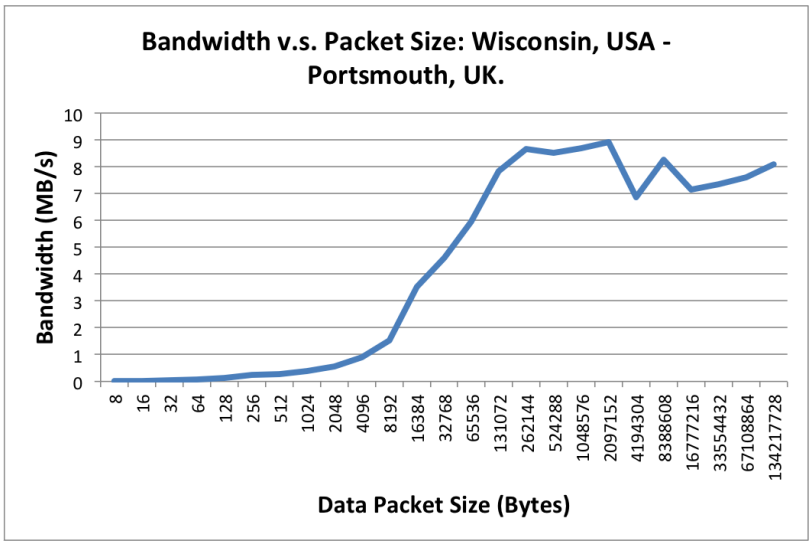


FIGURE 4.11: Achieved bandwidth for data streaming from WSRTI enabled synthetic application from Cray XC50 based in Wisconsin, USA to web browser on Macbook Pro based in Portsmouth, UK.

Figures 4.11 and 4.12 show WSRTI performance when streaming from a HPC system roughly 4000 miles away, and typical users of HPC centers in their own country, or even their own institution, may obtain higher bandwidths. Conversely, users in remote locations may indeed obtain much lower bandwidths. This can effect both the number of data packets that can be received per second, and the latency at which they are received. A real-world example of performance is discussed during implementation of a remote visualisation tool based on WSRTI in Section 4.3.

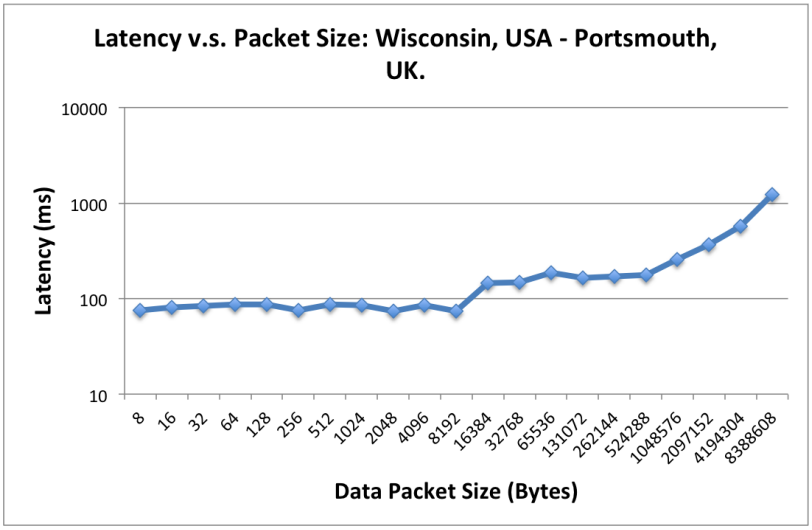


FIGURE 4.12: Packet latency for data streaming from WSRTI enabled synthetic application from Cray XC50 based in Wisconsin, USA to web browser on Macbook Pro based in Portsmouth, UK.

4.2.7 Discussion

The implemented approach represents a step toward smooth interoperability of high performance computing applications and the web, however there is still some distance to go in terms of support for this to become the norm. Particularly, a serious issue to consider is security and authentication when creating a direct connection between web applications and HPC applications. Future work should include streamlining the utilities to reduce the tax on HPC developers, particularly reducing the necessity for boilerplate code. One of the avenues to be explored is the use of a wrapper API to allow WSRTI to be used in a similar manner to popular in-situ visualisation libraries such as Paraview Catalyst and VisIt Libsim. There is also opportunity to further optimise the communication routines, especially for data streaming, for example including features such as automatic compression factor adjustment and tiled compression for multiplexed image streams. This should be combined with an investigation into the optimal approach to set up a WebSocket between HPC applications and Web clients from a security perspective, considering some of the approaches taken by existing frameworks such as Tapestry and ParaviewWeb.

4.3 A Remote, Interactive, Web-Based Visualisation Tool

One of the key motivating applications for WSRTI was Splotch. As outlined in the previous chapters, Splotch is designed for volume rendering of big, particle-based, astronomical data, an initial overview of code structure of Splotch can be found in Section 2.3.1. However, originally designed for remotely batch-rendering very large datasets, Splotch does not have interactive capabilities. The following subsections detail the extension of Splotch to remote web-based interactivity based on the WSRTI framework.

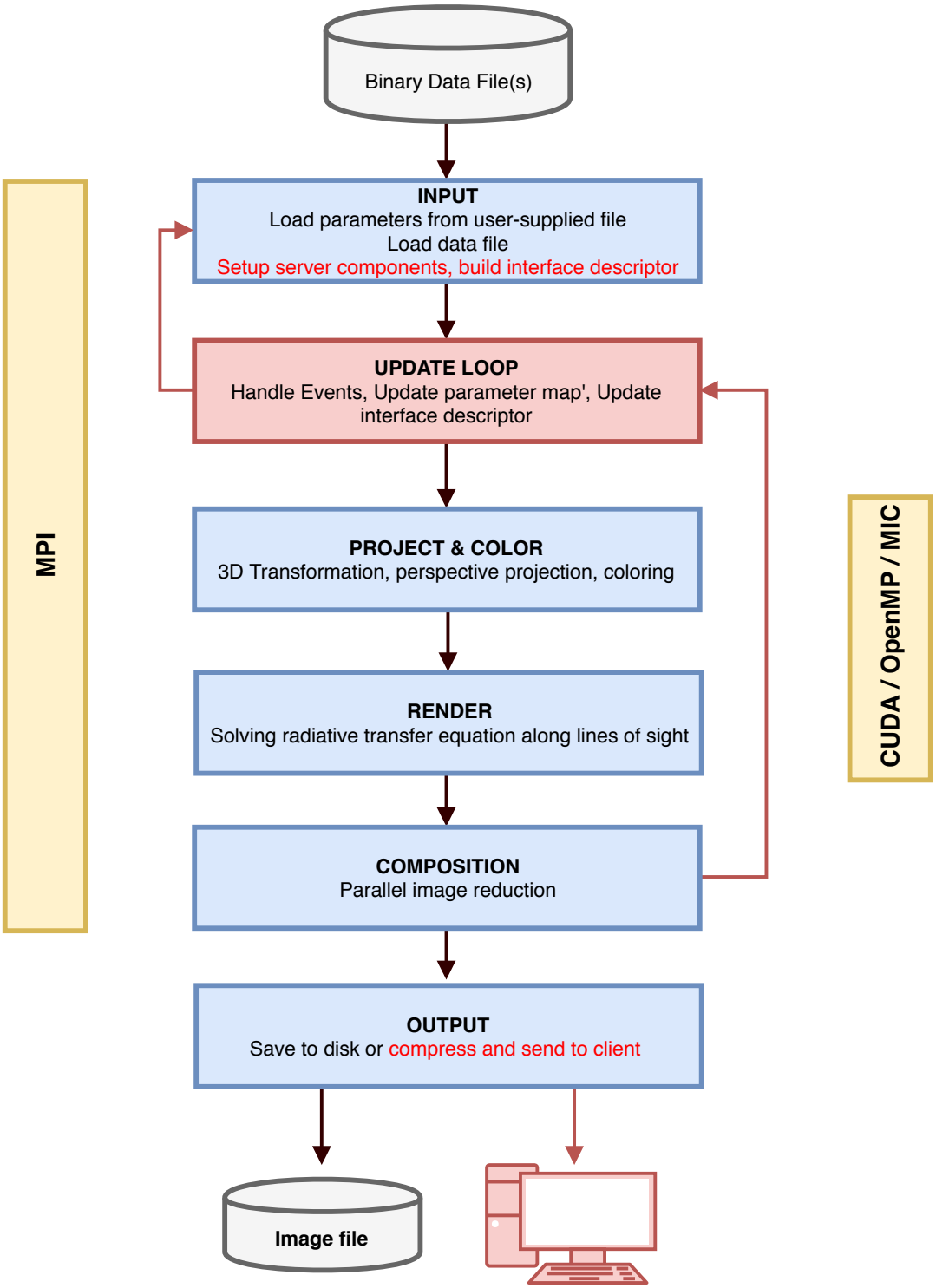


FIGURE 4.13: The Splotch algorithm, with changes for WSRTI interactivity highlighted in red.

4.3.1 Extensions for Remote Interaction with WSRTI

Detailed here are the steps taken to instrument Splotch with WSRTI, adding the minimum required functionality to support web-based remote and interactive visualisation, i.e. a bespoke implementation of the functions outlined in Figure 4.10. Figure 4.13 demonstrates the additions (in red) to the Splotch structure (originally shown in Figure 2.6), which are explained in the following subsections.

4.3.1.1 Instrumentation I: setup()

The first step in the instrumentation is to launch the WSRTI services for images and events, corresponding to the *data* and *auxiliary* websockets shown in Figure 4.5. For each service a callback, in the form of a C++ lambda function, is supplied; this is called whenever a message is received, and pushes the message into a queue for handling during *update()*. In addition, two threaded services are launched using C++ lambda functions to asynchronously send events, RPC commands, and images to the client (explained further in Section 4.3.1.3) .

Finally the default interface descriptor is built, which provides simple an input box to provide the path of a parameter file defining the input dataset and initial visualisation parameters. The C++ code to create this descriptor is demonstrated in Figure 4.9, with the generated JSON that will be sent to the client shown in Figure 4.8. These modifications are added to the initialisation stage, shown as *INPUT* in Figure 4.13.

4.3.1.2 Instrumentation II: update()

The second stage of instrumentation requires adding a loopback mechanism in the code, and inserting the *update()* function to handle scene updating, represented as the new *UPDATE LOOP* stage and arrow on the right hand side of Figure 4.13.

The update function handles:

Parallel Event Management:

Events received from the client are stored in a queue, and once each loop all existing events are handled here. Both interaction and generic events are handled here along with RPC calls (i.e. all the data flow towards the HPC application as shown in Figure 4.5). In an MPI parallel context, only the master rank interacts with the client, as such all events and RPCs are first forwarded via an MPI broadcast and handled in a data-parallel manner. Each event and RPC message is forwarded to a handler that takes the appropriate action, such as modifying scene parameters, updating local state, triggering response messages. A camera controller is introduced, to which camera interaction events, such as mouse clicks, are forwarded.

Scene updates:

To update the scene, local state must be checked for changes. Internal updates are performed, such as updating the frame rate tracker (used, for example, for smooth

camera movement). If the scene has changed, the remainder of the regular pipeline is followed to re-render the scene, otherwise we return to the event loop.

Interface updates:

The interface descriptor is updated based on any events that have occurred, in the initial implementation the available interface is minimal (as shown in Section 4.3.1.1), and expanded on in Section 4.3.2.

4.3.1.3 Instrumentation III: send_state_and_data()

In the simple initial implementation, the only state or data sent by Splotch is the current visualisation image. This is pushed to a synchronous queue (SyncQueue from WSRTI), which is then picked from the queue using C++ forwarding semantics (i.e. no buffer copies) by the image service launched in Section 4.3.1.1, compressed using the WSRTI JPEG compressor *TJPP*, and passed to the WebSocket server. Figure 4.14 demonstrates the minimal code used to set up the image sending service, followed by the send being triggered. In Splotch, the limiting factor on framerate is the data size; for large datasets (hundreds of Gigabytes and more), the frame-rate is expected to be low as compared to other real-time rendering (discussed further in Section 4.3.3). For this reason use only frame-by-frame compression is used, rather than video compression which would increase the latency for user interaction at low frame rates.

At this point, Splotch can be said to be both interactive, and remote, using the template web interface included with WSRTI. However, the motivation for this work is of course to create a tool that can be effectively used for remote visualisation, and as such the following section describes the first set of additional features implemented for the purpose of usability.

4.3.2 Building A Remote Visualisation Tool

To create a prototypical usable visualisation application, an additional set of features should be included. This consists of two stages, first a minimal set of additional features are added based the existing features of Splotch and common interactive visualisation tasks:

Loading and Reloading: One of the first requirements of an interactive visualisation tool is to interactively load, unload, and change the dataset. As a batch tool, the initial implementation of Splotch did not require to unload or reload data. An internal mechanism was built to check if a scene is already loaded when receiving an input parameter file. If so, the original scene is unloaded by clearing buffers and resetting all internal state to default, at which point a new set can be loaded based on the new parameter file.

```
// Launch the async services for image sending
void SplotchServer::launch_image_services()
{
  // Create image sending service
  auto imagesender = [this]() {
    tjpp::JPEGImage jpegImage;
    tjpp::TJCompressor comp;
    jpegImage.Reset(xres,yres,TJPF_RGB,TJSAMP_420,quality);
    while(server_active)
    {
      // This line blocks on Pop() until an image is pushed to the send queue
      jpegImage = comp.Compress(std::move(jpegImage), (const unsigned char*)
          &(ims_send_queue.Pop())[0], xres, yres, TJPF_RGB, TJSAMP_420, quality);
       // 'ims' is WSocketServer instance
       if(ims->ConnectedClients()>0)
           ims->Push(SerializeJPEGImage::PackDataOnlyWeb(jpegImage), false);
    }
  };
  imagesend_thread = std::thread(imagesender);
}

// Images are added to the queue by the master rank like so:
if(mpiMgr.master() && image_modified)
{
  ims_send_queue.Push(image_buffer);
}
```

FIGURE 4.14: The C++ code to set up a threaded image sending service. A C++ lambda function contains a continuous loop, which blocks on the Pop() function of a synchronous queue. When an image is inserted (as demonstrated in the lower part of the listing), it is compressed and passed to the WebSocket server to be forwarded to the client using C++ move semantics (i.e. no buffer copies).

Keeping the user informed: I/O is often a costly part of the visualisation process, especially for large datasets and bespoke scientific file formats that may not be optimised for parallel access. The user must be kept informed about the state of the application, especially during slow tasks like loading a large file. A loading thread is implemented, along with a set of routines to procedurally generate labelled images. When a user connects, while a dataset is loading, or if an error occurs while loading the data, the main thread can provide labelled images to keep the user informed of state whilst the loading thread handles I/O.

Client management: As the WebSocket servers listen to all traffic on a specific port, with a specific resource identification code, there is the possibility that multiple users could connect. Client traffic is distinguished at the WebSocket level, and so a client manager is added for multi-user support. The first client to connect becomes a master user, with full control of the visualisation and interaction, whilst later clients are observers. As such, a basic form of collaborative visualisation is supported. As highlighted in Section 4.2.4.4, the dual nature of the communication system means this could be further built upon for a peer to peer interactive collaboration visualisation tool.

Visual Parameter Modification: Beyond simple connection, data loading, and visual manipulation, one of the first tasks a user requires is to modify visual parameters. At this point a more extended interface is built, using the dynamic interface features described in Section 4.2.4.5. Each parameter available in Splotch is given a modifier interface element, either text input boxes, numeric sliders, and drop down list boxes (for example, the list of available colour palettes to). This includes:

- Display information (data set name, type, size)
- Camera sensitivity
- Image resolution
- Field specific visual parameters
- Type specific colour maps

This summarises the basic features included in the prototype; however, in order to define a set of features required to make the tool useful for astronomers, the appropriate approach is to work with such astronomers to ascertain user-defined requirements. The following chapter will describe the application of Splotch for theoretical astrophysical web portals, and the additional features developed in this context.

4.3.3 Performance

An interactive rendering application has higher expectations of performance than a batch application. As such, this section measures the interactive performance of

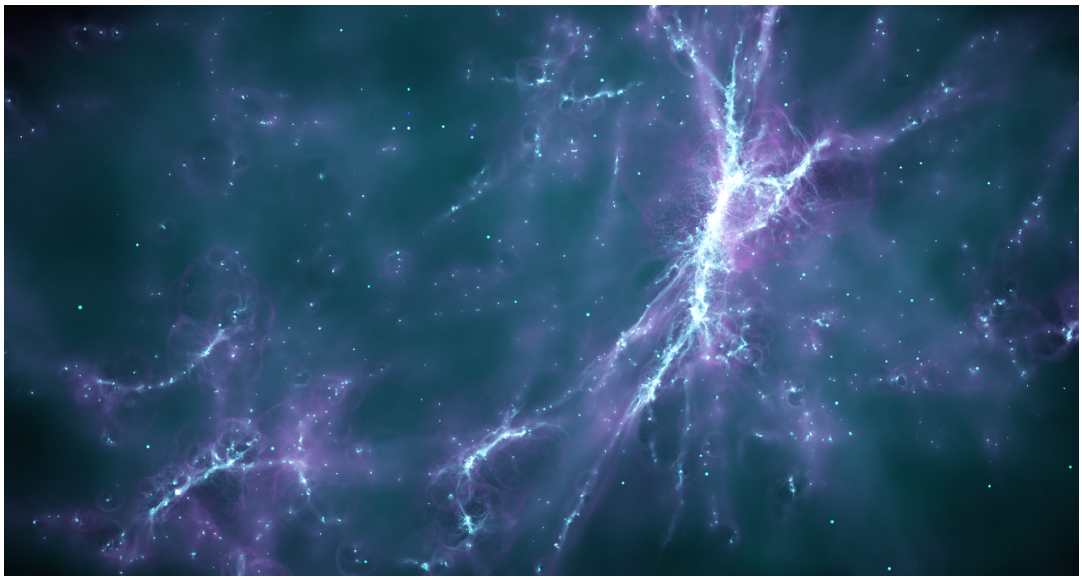


FIGURE 4.15: The GigaERIS snapshot (Mayer, et al., in preparation) used to test the Splotch interactive application, containing just over half a billion particles (553813391), or 18.5 GB after loading.

Splotch via strong scaling (i.e. varying the number of compute nodes whilst keeping the problem size static). Performance is presented in terms of *seconds per frame*, i.e. the time to render a single image, and presented alongside typical *frames per second* values, i.e. the number of frames rendered in a single second which is a common performance metric for graphical applications. Three hardware configurations are compared, Intel CPUs, NVIDIA GPUs, and Intel Xeon Phi Knights Landing (KNL). As a baseline, performance is compared against the frame rate expectations for large scale interactive scientific visualisation, where 5-10 FPS is typically considered a minimum (e.g. (Brownlee et al., 2012) (Hassan, Fluke, et al., 2013)).

The test scenario is a snapshot from state of the art galaxy formation simulation GigaERIS (Mayer, et al., in preparation), kindly provided by Lucio Mayer of the Institute for Computational Science, University of Zurich. In the scene, star and gas particles are shown (the encompassing field of dark matter is disregarded), resulting in just over half a billion particles (553813391), or 18.5 GB after loading and converting to Splotch internal representation. Sustained performance is reported, averaged over 100 frames, and strong scaling is measured up to 64 nodes where possible.

The three test configurations are each varieties of Cray XC systems, exploiting: (1) Intel CPUs, (2) NVIDIA GPUs, and (3) Intel Xeon Phi Knights Landing. The hardware configurations are as follows:

- **(1)** Up to 64 nodes; dual socket Intel Broadwell E-5299 22 core CPUs (2.20 GHz); 1 MPI task per socket (up to 128 tasks); 22 OpenMP threads per MPI task (up to 2816 threads).
- **(2)** Up to 32 nodes; 16 Tesla K40, 16 Tesla K20X, 1 MPI task per node. (K40 preferred for tests <= 16 nodes).

- **(3)** Up to 64 nodes; Intel Xeon Phi 7250 Knights Landing 68 core CPU (1.40 GHz); 4 MPI tasks per node (up to 512 tasks); 17 threads per MPI task (up to 8704 threads).

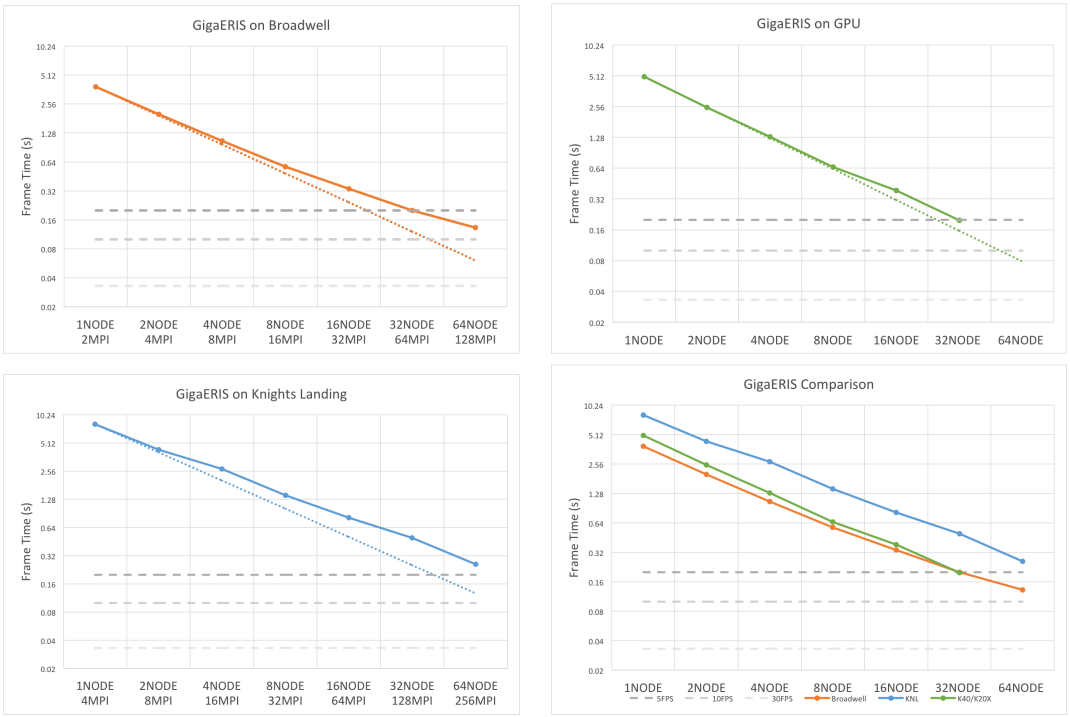


FIGURE 4.16: Each of the three test configurations is displayed separately, Intel Broadwell *(top left)*, NVIDIA GPUs *(top right)*, and Intel Knights Landing *(bottom left)*. Scaling results are overlaid with common performance targets for interactivity: 5, 10, and 30 frames per second shown as grey dotted lines from top to bottom respectively. Finally the configurations are compared *(bottom right)*.

Figure 4.16 shows performance results up to 64 nodes for Broadwell, GPU, and KNL. Initial setup costs such as I/O and memory allocation are discounted, as they are one-time expenditures and not relevant to the interactive performance. Each result (sold coloured line) is first shown individually compared with linear performance (dotted coloured line), and finally the three implementations are compared to each other. Each graph is marked with three grey dotted lines indicating the performance necessary to achieve 5 FPS, 10FPS and 30 FPS. It should be noted that for low node counts the GPU tests are subject to the additional overhead of block transferring data into device memory, which for K40 is 12GB. Furthermore, the Knights Landing tests (run in native mode) do not have architecture specific optimisations applied beyond appropriate flags at compilation time; as such, this should not be taken as a fair comparison of the KNL and GPU architectures, considering the relative time spent on architecture-specific optimisation.

Finally, Splotch is already a high performance code designed for large data and HPC systems, and these initial performance tests show that Splotch is capable of achieving interactivity; however, further modifications are required for interactive performance that were not necessary in the original batch mode. As a first step,

obvious performance areas have been addressed, such as reducing memory allocations through buffer reuse, however future work will focus on micro-optimisation to achieve better interactive performance.

4.4 Summary

This chapter addresses the objective **O.3**: *'Explore a general approach to addressing web-based interactive high performance visualisation'*.

A novel classification scheme for remote high performance applications is presented (Section 4.2.2). This is followed by a review of the state of the art in remote, interactive, high performance visualisation software in the context of the presented classification scheme, identifying a lack of a general approach to connect high performance software and web applications (Section 4.2.3). To address this lack, a general purpose approach to support the interoperability between interactive high performance computing applications and web environments is presented (Section 4.2.4). To demonstrate this approach in the context of high performance visualisation, the formerly-batch software Splotch is transformed into interactive and remote visualisation tool (Section 4.3). The performance of this tool is briefly evaluated across three high performance architectures, demonstrating the feasibility of real time interaction and parallel scalability (Section 4.3.3).

In view of the question posed: **Q.3**: *'How can modern astronomical tools exploit remote and interactive high performance visualisation on the web?'*, this chapter addresses the problem of interoperability between interactive HPC applications and web environments and exploits a solution for remote and interactive high performance visualisation on the web to demonstrate a general approach for remote and interactive high performance visualisation tools to be exploited in the context of modern, web-based, astronomical tools.

Chapter 5

Visualisation for Theoretical Virtual Observatories

This chapter aims to answer the question **Q.4**: *'How can remote high performance visualisation facilitate access and analysis of large astronomical datasets on the web?'*, by addressing the corresponding objective **O.4**: *'Discover if and how high performance visualisation can support astronomers in typical web-based access and analysis scenarios'*. The chapter begins with a brief introduction to access and analysis of theoretical data via web environments for astronomy, expanding on Section 1.3.4. The current state of the art in 3D visualization within web-based astronomy data portals is reviewed, followed by a discussion of the general process of data access within such portals and the technical challenges involved (Section 5.2). A series of use cases are identified and analysed to extract a set of general requirements for interactive visual discovery, which are then met via a prototype tool building on the work of previous chapters (Section 5.3). The utility of this approach is then demonstrated by reconstruction of the example use cases (Section 5.4), with discussion presented (Section 5.5). The chapter concludes with a brief summary of research contributions (Section 5.6).

5.1 Web Environments for Astronomy

Section 1.3.4 introduced the growing popularity of web services for serving the data needs of the astronomical community. Some notable examples include the extensive services incorporated into the Strasbourg Astronomical Data Center (CDS)[1], and projects related to the International Virtual Observatory Alliance (IVOA)[2] such as the German Astrophysical Virtual Observatory (GAVO)[3], and the All-Sky Virtual Observatory (ASVO)[4]. These are on-line umbrella services that have historically hosted very large repositories of observational images and surveys in standardized formats, alongside web-based tools to assist in finding, identifying, and extracting data from the archives, for example Vizier (Ochsenbein, Bauer, and Marcout, 2000) and the Aladin Lite Sky Atlas (Boch and Fernique, 2014) hosted at the CDS.

[1] http://cdsweb.u-strasbg.fr/
[2] http://www.ivoa.net/
[3] http://www.g-vo.org/pmwiki/Main/HomePage
[4] http://www.asvo.org.au/

In the case of theoretical data, such as large cosmological simulation outputs, a public data release will often link to an online database where users can download some or all of the data. For example, the Millennium Simulation Data Archive (Lemson and Virgo Consortium, 2006) is hosted through GAVO and accessible via SQL query, and the outputs of the EAGLE simulation (McAlpine et al., 2016) are hosted and query-able in a similar manner at the Institute for Computational Cosmology of Durham University. While this is undoubtedly beneficial for scientists, access to the raw data is not the only solution that can be offered by data portals, and is sometimes the least convenient. The raw size of data releases means a potentially non-trivial process of identifying and extracting useful data can become prohibitive without dedicated computing resources.

As introduced in Section 1.3.4, there is now emerging a more advanced type of web portal for theory data that not only provides access to the raw outputs of simulation, but includes tools to assist the astronomer in exploring the data as well as creating or accessing derived datasets that can be more directly useful for their science case. An early example of this type of repository is the Mock Map facility (Blaizot, Wadadekar, et al., 2005), an online tool that allows users to access mock galaxy catalogues generated via the GALICS semi-analytic model (Blaizot, Guiderdoni, et al., 2004). More recent examples (see Section 5.2.1) are also supported by powerful hardware back-ends such as High Performance Computing (HPC) systems to support more complex tools for selection, customization and generation of derived data.

Astrophysical simulations can have extremely high computational and memory requirements, requiring large scale solutions provided by HPC facilities. For example, the raw data of the flagship Millennium run consisted of over 10 billion particles and was executed on the Max Planck Societies principle HPC system for over a month, with the final output requiring roughly 25 TB of storage. These types of data products are far too large to fit in the memory of a personal workstation and require large scale computing resources to host and process. Due to the fact subsequent processing and analysis requires HPC as well, it is a natural consequence that web portals for these data products are also being supported by HPC systems.

This chapter aims to demonstrate how integrating interactive 3D visualization with existing platforms can support users of such platforms in accessing, filtering, exploring, and extracting theory data. The aim is to ensure that larger datasets and demanding processing tasks remain within the web portal, while the user only downloads and processes the minimum amount of data they require. Demanding processing tasks can then rely on the support of HPC resources to attain the performance necessary to visualize and interact in real-time.

5.2 Web Visualization for Astronomical Portals

5.2.1 The Web Observatories

There is a new type of theoretical astronomical web portal emerging, exploiting advances in modern web technologies and powered by high performance back ends. This section focuses on the visualization capabilities of those characterized by hosting large theoretical data for public access with the support of high performance computing resources, summarized in Table 5.1. Three types of visualization capability are noted:

2D: Two dimensional plotting capabilities such as histograms, line graphs, scatterplots.

3D: Three dimensional plotting capabilities such as maps and 3D renderings.

VObs: Virtual observations generated by scientific mock imaging packages.

For each type of visualization the availability type is specified, either *user-generated* or *pre-computed*, which distinguishes respectively whether the user can generate their own new visualizations or simply browse a selection of existing visualizations.

Furthermore, the type of interaction available is specified: *static* in the case of simple images; *2D interaction* in the case of Zoomify-like[5] navigable images; and *3D interaction* for 3D rotate and zoom navigation such as found in traditional interactive 3D visualization packages.

TABLE 5.1: Visualization in Theoretical Astronomy Portals

Portal	Visualization	Availability	Interaction
(Overzier et al., 2013)	VObs	pre-computed	2D interaction
(Chard et al., 2014)	2D	user-generated	static
	VObs	user-generated	static
	3D	user-generated	3D interaction
(Nelson et al., 2015)	VObs	pre-computed	static
	3D	pre-computed	2D interaction
(Bernyk et al., 2016)	VObs	user-generated	static
(Ragagnin et al., 2016)	2D	user-generated	static
	VObs	user-generated	static
	3D	pre-computed	2D interaction
(Carretero et al., 2017)	2D	user-generated	static

The Millenium Run and later simulations are housed in the Millenium Run Database, an SQL-queryable database for the original data outputs of the simulations (Lemson and Virgo Consortium, 2006). This is combined with the Millenium Run Observatory (Overzier et al., 2013) hosting lightcone catalogues with virtual observations in many different configurations and other associated data, all of which can be traced back to the data products hosted in the Millenium Run Database.

[5] http://www.zoomify.com/

PDACS (Chard et al., 2014) is a web portal and science gateway built upon the GALAXY workflow engine for life sciences research[6], re-purposing the engine as a platform for data access and analysis for a suite of cosmological data products forming the Coyote universe (Lawrence et al., 2010). The platform is supported by NERSCs computing infrastructure and ANLs scientific cloud. It incorporates a variety of data analysis services, including ParaviewWeb (discussed further in Section 5.2.3) integration that enables interactive 3D visualization(Madduri et al., 2015).

The Illustris project (Vogelsberger et al., 2014) released a web portal[7] along with their public data release (Nelson et al., 2015) which hosts query-able data produced by the suite of Illustris simulations along with various tools for data searching and exploration. The site is supported by a cluster back-end and includes *The Explorer* for 2D exploration of Illustris visualizations, along with the *Galaxy Observatory* which provides mock images of galaxies at z=0 where stellar mocks have been built. While visualizations are currently pre-computed, the authors indicate that future promising directions include providing tools and resources for users to compute their own arbitrary results from the hosted data.

Bernyk et al. (2016) describe the Theoretical Astrophysical Observatory. TAO is an online virtual observatory providing access to mock extragalactic survey data generated by running complex semi-analytic galaxy formation models on the output of large N-body cosmological simulations. TAO provides a variety of science modules for users to generate and extract data useful for their specific science case, supported by a supercomputing system as a back-end. In terms of visualization, users can apply a science module to generate custom mock image observations. TAO is an example-case and motivation of the presented effort toward incorporation of interactive 3D visualization to virtual observatories, and discussed further in Section 5.4.

The Cosmological Web Portal (Ragagnin et al., 2016) is an online web service for hydrodynamical, cosmological simulations. Supported computationally by LRZ and C2PAP, the portal provides services to browse and subset Magneticum data[8] as well as generate various 2D maps, virtual observations and spectra. Visualizations in 3D are pre-computed via the Splotch software. These services create jobs on the underlying HPC system in order to satisfy user requests, and so allow users to not only explore existing raw and derived data, but generate further results to suit their own needs.

CosmoHub (Carretero et al., 2017) is a web application built upon Apache Hive cloud services to host large survey data, allowing users to generate their own catalogues or download existing catalogues and supplementary files. A Python visualisation tool exploiting Plot.ly[9] is integrated for 1d plotting and 2D visualisation, which can be generated on demand.

[6] https://galaxyproject.org/
[7] http://www.illustris-project.org/
[8] http://www.magneticum.org/
[9] https://plot.ly/

As can be seen from Table 5.1, in general it is not common to include interactive 3D visualization for astronomical web portals. The exception, Chard et al. (2014), is the most advanced platform, built upon an existing workflow engine and customized for cosmological workflows. A key difference between PDACS and the other portals presented is that it is a workflow engine for cosmology, rather than a public access repository. For example, in order to use PDACS a user must have an associated account with computing allocation.

A strong factor in the lack of 3D vizualization for web portals is the computational expense. Large theoretical datasets require HPC resources for visualization, however achieving interactivity is not trivial on HPC systems due to both the computational expense and difficulties integrating web and HPC environments (the focus of the previous Chapter 4). Astronomy focused technical challenges are further discussed in Section 5.2.3.

5.2.2 Visualization as part of the Data Access Workflow

Visualization is an essential factor in effective data exploration, and it can also support the data access workflow. The workflow in Figure 5.2 is a general representation of the steps a scientist, or user, would take to access data within a web platform. The user will start by defining an initial dataset, and apply post-processing to build a derived dataset. This may include an extraction process in which the user applies filters to constrain data properties and extract a specific data subset (see example interface from the TAO platform in Figure 5.1), as well as applying transformations via science modules made available to the user. Finally they may download the resultant dataset(s).

In current web repositories without on-demand analysis, the user must download their data before performing some initial data processing to confirm it is as required. This may be done with a variety of analysis techniques, often including local visualization. However, if after some analysis and exploration it is clear that the data is not exactly as required, the user may need to return to the web portal and begin their process again. This feedback loop represents a trial and error process, an unnecessary barrier that can be time consuming and interrupt the users workflow. It is highlighted in Figure 5.2 as the solid-line green box on the left side of the figure.

Conversely, the dotted red box in Figure 5.2 represents the inclusion of in-situ and on-demand analysis techniques. In this case, in-situ refers to the preliminary analysis being performed within the portal where the data is situated, while on-demand refers to the use of scheduled HPC resources to fulfill analysis task requirements. In-situ and on-demand visualization allows the user a quick view of their dataset during the filtering and transformation process, helping to identify areas of interest or visible errors before downloading the data.

Furthermore, an interactive visualization tool that includes some built-in means of manipulation and filtering can enhance this process by allowing the feedback loop between visual examination and filter application to take place interactively.

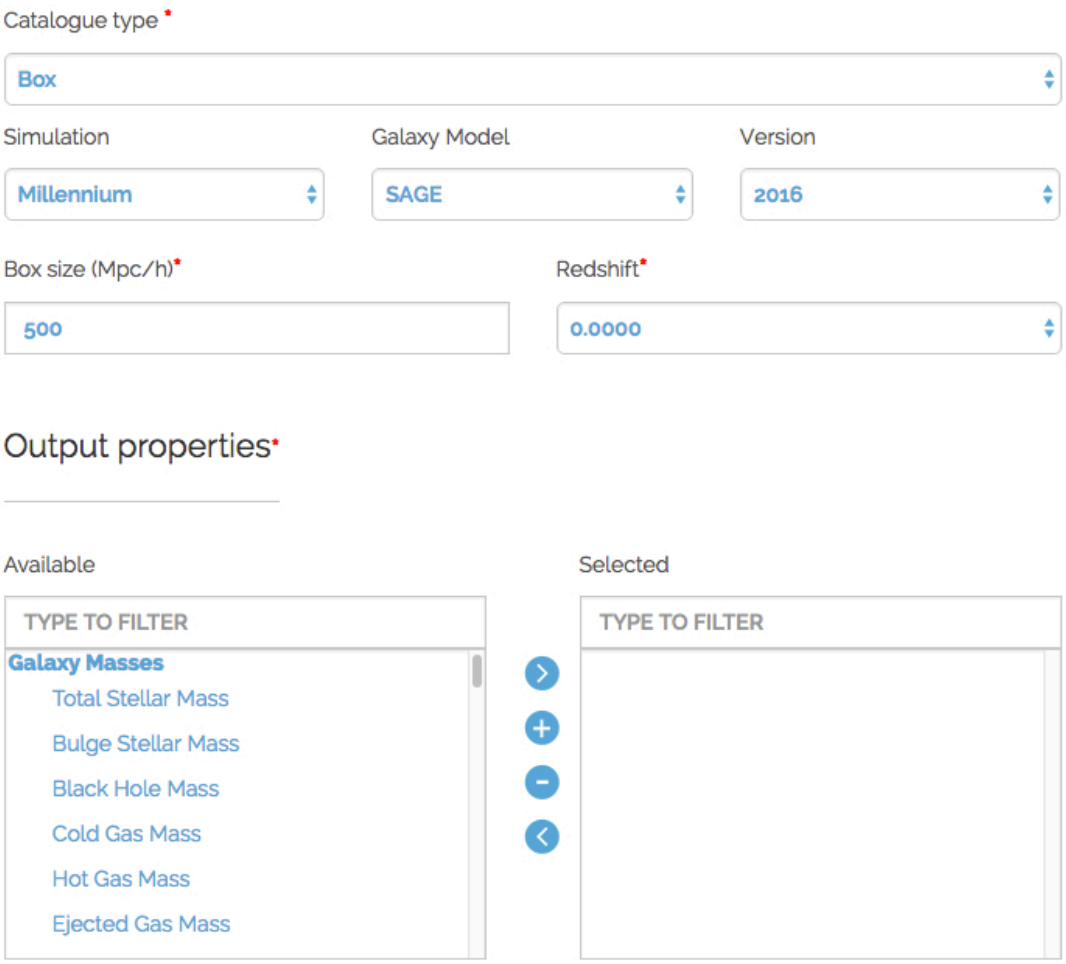


FIGURE 5.1: Example section of the interface for specifying and extracting data from the Theoretical Astrophysical Observatory.

This enables a more intuitive exploration and filtering process where the user can add and remove experimental filters as necessary with real-time visual feedback. The ultimate goal is to provide the scientist with a clearer understanding of their data before leaving the portal to apply their own methods.

5.2.3 Technical Challenges for Web Visualization in Astronomy

In order to exploit scientific visualization on the web, visualization tools must be able to communicate with the ecosystem of software that makes up modern web applications. For example, current browsers come equipped with powerful engines for Javascript, which has become the de-facto language of the web. As such, the various approaches for 3D rendering in a web browser are predominantly Javascript libraries based on the WebGL graphics API (for a thorough review of these technologies the reader is referred to Evans et al. (2014)). The most notable web-only approach to 3D scientific visualization is the currently ongoing effort to port a subset of the Visualization Toolkit library[10] to Javascript as vtk.js[11]. Further approaches can be seen in

[10] https://www.vtk.org/
[11] https://kitware.github.io/vtk-js/

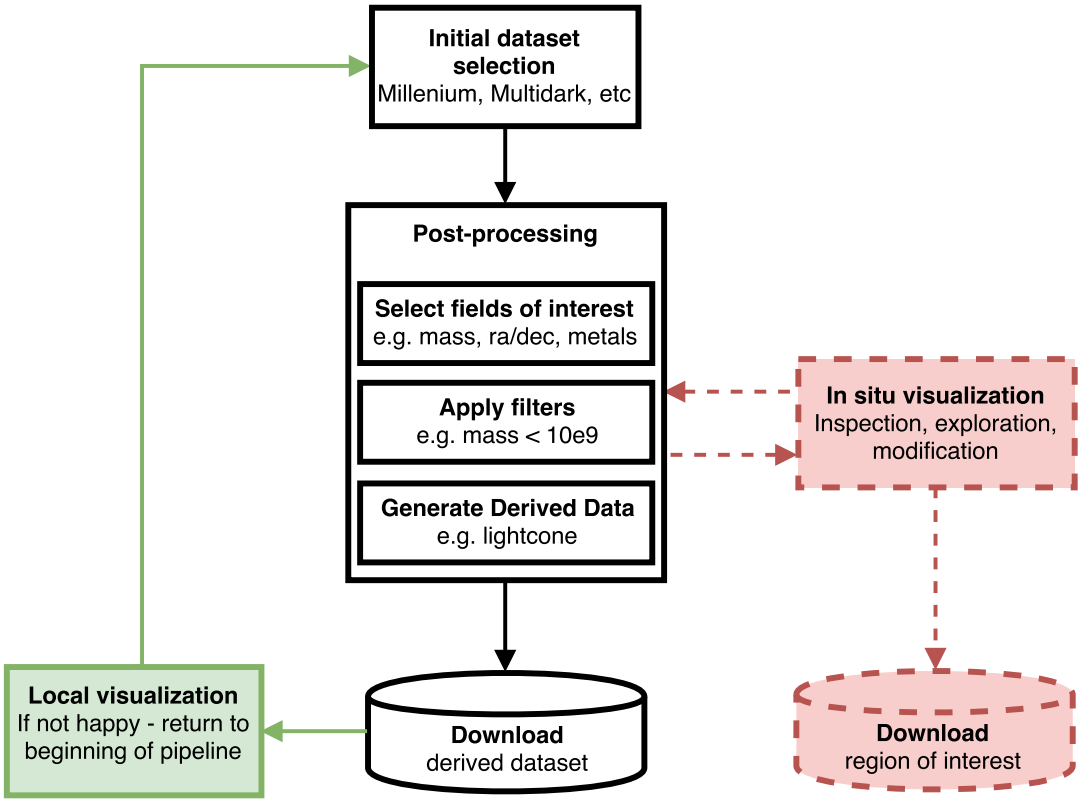


FIGURE 5.2: The process for a user to access theory data from a web portal. The feedback loop in green highlights the current process for data inspection, while the feedback loop in red shows the proposed process.

the recent state of the art review of web-based visualization techniques (Mwalongo et al., 2016).

Whilst web-only approaches to visualization would be ideal, two of the most challenging technical aspects of 3D visualization for astronomy are the computational and memory requirements. Theoretical astronomical datasets are often orders of magnitude larger than one could store and process on a home desktop or laptop. For example, the semi analytic galaxies currently stored on TAO[12] are built from N-Body simulations ranging up to tens of billions of particles, while at the extreme scale modern day simulations are advancing even further with a recent run of the PKDGRAV cosmological code exploiting two trillion particles and requiring roughly 124 terabytes of storage (Potter, Stadel, and Teyssier, 2017). With current web-only visualization tools it is not possible to visualize data of this size interactively in 3D solely via web browser. Instead, as seen in Mwalongo et al. (2016), it is possible to perform remote 3D rendering via a powerful back-end computing system connected to a web based client in a client-server architecture.

The concept of performing large data processing as physically close as possible to the storage is coined *moving computing to the data*. Transferring large amounts of data to the client for rendering is often impractical, if not simply impossible, due to data size and network transport. However, sending compressed images is often a much easier task and it is well within the capabilities of modern browsers to receive and display these at high speed. In this context it is common to use a client-server architecture that performs compute tasks on a server close to the data and streams images or video to a client, reducing large data transfers over wide-area networks such as the internet.

There are two notable solutions for client-server web based visualization for astronomy. Firstly, as already mentioned (see Section 4.2.3 of the previous chapter), the well-known general purpose scientific visualization tool Paraview (Ahrens, Geveci, and Law, 2005) and associated web-toolkit ParaviewWeb[13]. The web-toolkit allows developers to integrate a web-client for the Paraview server into their application. While the web framework is relatively recent it is the most advanced tool currently for web based scientific visualization, with options from web-only visualization to a full client-server mode with a Paraview or VTK (Schroeder, Martin, and Lorensen, 2006) backend.

VisIVO is a suite of visualization software targeting the virtual observatory environment (Comparato et al., 2007) (Becciani et al., 2010). The suite includes a client-server tool supporting grid computing systems, which can integrate with a web-based science gateway tool built around grid-computing middle-ware workflow tools(Sciacca, Bandieramonte, et al., 2013). The underlying viewer can exploit both VTK and Splotch as rendering engines. VisIVO is currently under development to extend capability for visual analytics (Sciacca, Becciani, et al., 2015), and is

[12] https://wiki.asvo.org.au/display/TAOC/Available+Data+Sets
[13] https://github.com/Kitware/paraviewweb

currently the most VO-compliant scientific visualization tool.

A more recent concept that has gathered momentum is the use of cloud technologies for high performance processing, including visualization. NVIDIA offer a GPU Cloud service that can provide high performance visualization capabilities on cloud based GPU hardware with minimum setup and installation[14]. There is not yet much progress toward utilizing a service such as this for visualization in a virtual observatory setting. A key hurdle to this approach is the location of the source data, usually already stored in a HPC center. To use cloud based visualization services the web platform must also store the data within the cloud, which can be costly. However, there is definitely potential in this area, and cloud-like approaches exploiting cluster computing, such as Tapestry (introduced in Section 4.2.3 of the previous chapter), are starting to gain traction.

In the remainder of this chapter, a new approach will be demonstrated that is tailored for astronomers, and can exploit high performance resources while being utilised from a web environment.

5.3 Interactive Web-based Visualisation via the Theoretical Astrophysical Observatory

The Theoretical Astrophysical Observatory (introduced in Section 5.2) is a web platform hosting data from multiple cosmological dark matter numerical simulations and galaxy formation models. TAO includes various science modules for users to apply to their data, this includes extracting a light cone with parameters that match particular survey geometry, or generating custom mock images through SkyMaker (Bertin, 2009). A node of the ASVO project[15], TAO is supported by the Center for Astrophysics and Supercomputing (CAS) at Swinburne University of Technology, Melbourne, Australia. In addition to support and administration, CAS provides use of the Swinburne HPC cluster as a high performance back-end to support the cloud-based platform. As a web platform for theoretical data supported by HPC resources, and built in a modular way to enable integration of new science modules, TAO is an ideal example platform to motivate and demonstrate the utility of 3D visualization in this context.

5.3.1 Use Cases

Through a series of informal discussions with the astronomers running TAO, two use cases were identified that could benefit from a process of interactive visual discovery. These use cases exemplify the more general process described in Section 5.2.2 (and illustrated in Figure 5.2).

(1) Exploring the environments of recently merged galaxies

[14]https://www.nvidia.com/en-us/gpu-cloud/hpc-visualization-containers/
[15]http://www.asvo.org.au/

One example of the power of interactive data visualisation for a service like TAO is as a tool to find the large-scale environments of recently merged galaxies. Galaxy-galaxy collisions are common and can have dramatic effects on the resulting galaxy's properties. Just where and when these occur is a matter of active study by the community. After creating a mock survey universe in TAO, interactive visualisation exploiting interactive filtering would allow the user to draw out recently merged systems (or even those about to merge) so that their properties can be studied and cross-correlated with the broader environments in which they live (void, filament, group or cluster). Such information could then be used for mock imaging or for additional processing once downloaded locally, and directly compared with equivalent observational data sets. In a recent study utilising TAO, Penny et al. (2015) sub-sampled galaxies from the Galaxies and Mass Assembly survey (Driver et al., 2009) to study bright galaxies found in voids. One question posed by the authors is whether the stellar mass of these galaxies is primarily attained through star formation or mergers. The study made use of theoretical data in the form of a galaxy catalogue produced by the semi-analytic galaxy model SAGE (Croton et al., 2016) run upon the Millennium N-body simulation, which includes information unavailable in observational data such as the time since last major merger. The authors extracted these galaxies from the TAO archive via a series of filters in the web interface, followed by further neighbour-based local environment filtering, and presented visualization comparisons (e.g. Figure 12 of Penny et al. (2015)). Such an extraction process could be enhanced through interactive visualisation, and serves as an ideal use case for building requirements.

(2) Target Selection in the Large Scale Galaxy Distribution

A common scenario an astronomer faces is to find a specific galaxy population for analysis amongst the wider distribution of galaxies spread across the Universe. For example, astronomers commonly study the physics of massive galaxy formation inside galaxy clusters, which host all kinds of interesting phenomena such as red-and-dead elliptical galaxies and active radio galaxies. Beyond galaxy evolution, such clusters are also important probes of the cosmology of the Universe and can be the focus of highly sophisticated observing campaigns (e.g. eRosita (Merloni et al., 2012)). Using TAO one can build a mock universe that includes many millions of galaxies out to great distances. Currently, to sub-select from this larger population and identify e.g. clusters, the data needs to be first downloaded and processed by the user locally with their own code. Any subsequent imaging within TAO then requires the user to run a new job using this new information, duplicating effort and data. This scenario may be streamlined and enhanced through a process of interactive visual exploration and discovery in which the user can apply a series of filters to subset the galaxy distribution, and then progress to the application of general functions to modify and combine data fields.

5.3.2 Requirements

The use cases of Section 5.3.1 were analyzed in collaboration with the astronomers running TAO to gather a set of general requirements for interactive visual discovery. These requirements will define the features necessary in a web-based visualization tool and are split into four categories. *Data* requirements are those pertaining the the handling and management of domain specific datasets. *Client interaction* requirements are those influencing the usability of the application. *Knowledge discovery* requirements allow the astronomers to explore and investigate their data. *Knowledge extraction* requirements allow the astronomers to extract and save any data that may support the knowledge gained during the visualisation session.

Data Requirements

R.1 Support for galaxy catalogues

The visualisation tool must support galaxy catalogue datasets. This requires the loading of a series of data fields from a HDF5[16] format file. At minimum there will exist three dimensions specifying position in 3D cartesian coordinates, beyond this there will be an unspecified number of labeled data fields. Each data field found in the file must be listed by label and available for visualisation.

Client interaction requirements

R.2 Interactive file management

The visualisation tool must support interactive loading and unloading of files, to change data files without reloading the client. Furthermore, the user must be able to interactively choose which of the available fields is to be used for each visualisation quantity.

R.3 State management

The user must be able to load and save state such that an interactive visualisation session can be stopped and resumed exactly where the user left off at a later date. This may involve a save file stored locally.

Knowledge discovery requirements

R.4 Data colouring

The user must be able to colour each data field individually, and swap both field currently being viewed and color map currently being used interactively.

R.5 Data filtering

The user must be able to apply filters to the data, selectively viewing data subsets based on user-specified maximum and minimum values, which must be interactively modifiable.

[16] https://www.hdfgroup.org/solutions/hdf5/

R.6 Data transformation

The user must be able to apply transformation functions to data fields to more easily comprehend the data. At a minimum, it must be possible to apply an in-built logarithmic transformation to simplify the viewing of high dynamic range astronomical datasets

R.7 Data combination

It must be possible to apply general functions to data fields to generate new data fields for visualisation, using other existing data fields as operands.

Knowledge extraction requirements

R.8 Image saving

The user must be able to take a snapshot of the current visualisation output, and download this as an image file.

R.9 File saving

The user must be able to generate a new HDF5 formatted galaxy catalogue file, including any derived data fields generated during visualisation.

R.10 Meta-data

The user must be able to access meta data for saved images and files that documents the visualisation process and original datasets required to reproduce the associated image and/or saved file.

5.3.3 Addressing the Requirements within Splotch

5.3.3.1 Data Requirements

Simulation data is rich in complexity; a wide range of fields can typically be stored, representing many different data properties both physical and non physical, and data structures ranging from hierarchical structured grids to scattered point data. Astronomical file formats often reflect this complexity, with many large simulation codes including bespoke file formats (e.g. Gadget (Springel, 2005), RAMSES (Teyssier, 2002), Enzo (Bryan and Norman, 1997), to name just a few). Splotch addresses this with a generic file interface implemented for many of the popular astrophysical file formats. Each implementation extracts data from the file, as specified by an input parameter file, fills a Splotch native particle structure and discards any contextual information relating to data fields. This has been an effective approach for batch rendering; however, an interactive visualisation tool must give the user the ability to retain data context while visualising.

HDF5 is a popular hierarchical data format and accompanying API supporting parallel I/O, and is growing in popularity in astronomy; within TAO, all galaxy catalogues are stored in the HDF5 format. To address requirement **R.1**, a new file

reader was implemented to support HDF5 files. The reader first analyses the file header and stores contextual information such as number of fields present, a list of field names, number of galaxies, and other dataset specific information. Data fields existing in the file can be queried by the stored header information, allowing to support any combination of labelled data fields.

5.3.3.2 Client Interaction Requirements

The visualisation client must allow the user to interactively explore the data, swapping fields and modifying data characteristics. As mentioned in Section 4.3.2, support for interactively unloading and loading whole data files was added to Splotch. However, to address requirement **R.2**, more comprehensive file interaction support is necessary. A new generic file interface was implemented to address this issue. The new interface supports querying the fields available in a file, loading and unloading fields by label, and is supported by a new data management module in Splotch to manage state for interactive file manipulation.

The interface consists of the following functions:

read_info Read any contextual information available in the file, such as size and available fields.

read Read the portion of the file as specified by the provided parameters, returning native particle list.

read_field Read a specific field to local storage.

write_filtered Write a filtered dataset based on the specified filter map.

reset Reset all local state and remove local data.

particle_fields List fields actively utilised for particles.

available_fields List fields available in dataset.

Along with additional utility functions for opening and closing the file.

Requirement **R.3** necessitates the ability to save, and reload, the current state. To support this, a state management module was introduced to Splotch. The new module can capture the entire application state on request and store in a JSON state file to be downloaded. Upon provision of a previously downloaded JSON state file, the state module can reload the required data file and restore application state to continue a visualisation session where it was previously left. Further discussion on the state file can be found in Section 5.3.3.4.

5.3.3.3 Knowledge Discovery Requirements

Knowledge discovery requirements are centered on the concept of *quantitative visualisation* which allows the user to directly tie the numerics of their data to the visual

output of a visualization, inferring quantitative information from visual presentation (Peskin et al., 1991). Each of the requirements in this section aim to provide the means to perform quantitative data processing as part of the visual interaction process.

Requirement **R.4** is already partially supported as described in4.3.2, with type and field specific colour maps interactively selectable. With the addition of interactive field swapping as discussed in Section 5.3.3.2, this support was extended such that colour maps are automatically updated when the visualised field is changed.

To address requirements **R.5**, **R.6**, and **R.7**, a data filtering menu has been implemented, allowing the user to perform a variety of quantitative tasks on their dataset. Three types of filter are supported:

Clipping filters, addressing requirement **R.5**, allow the user to subsample their data by inputting ranges of any existing field in the data (including those not currently being used in the visualization). An example of this could be clipping a lightcone geometrically in Right Ascension and Declination, followed by applying a mass cutoff to show only very high mass galaxies.

Arithmetic filters, addressing requirement **R.6**, provide a way of scaling or otherwise arithmetically modifying any existing field of the data. An example of this could be applying a logarithmic filter to the visualized quantity to better distinguish fields across a high dynamic range, or scaling data quantities for use e.g. in combinatory filters.

Combinatory filters, addressing requirement **R.7**, allow the combination of multiple fields of the data to create derived fields for visualization. A user can provide data fields as left and right hand sides of a statement, and pick a combinatory function to apply such as adding or subtracting. This can, for example, be used to subtract colour bands from one another in a galaxy survey, the differences of which can point to patterns in the data such as concentrations of old vs young stars or be used as a proxy for redshift. Combinatory filters provide a strong opportunity for knowledge discovery by allowing users to generate new datasets during visualisation.

Filters can be stacked on top of one another which allows a user to interactively build e.g. advanced selection cuts for a galaxy catalogue. An example of the potential applications of combinatory and arithmetic cuts can be seen in Eisenstein et al. (2001), where multiple colour bands are combined and various scale factors applied to create derived fields that are used to select luminous red galaxy targets from the Sloan Digital Sky Survey. The process of quantitative visualization through data filtering allows a user to interactively experiment with selection cuts such as these, with interactive visual feedback from their data to aid the scientific process.

5.3.3.4 Knowledge Extraction Requirements

Knowledge extraction requirements focus on supporting the user in extracting knowledge and supporting data from the visualisation session. This includes visualised imagery, derived datasets, and contextual meta-data.

```
{
    "Filename": "/Users/tims/Data/TAO/tao_tdykes_catalogue_2031/0/tao.2031.0.hdf5",
    "Type": "HDF5",
    "Original Points": 136298,
    "Filters": {
        "Filter 1": {
            "Type": "COMB_FIELD",
            "Field": "SDSS_g_Absolute"
        },
        "Filter -1": {
            "Type": "NORM",
            "Field": "SDSS_g_Absolute"
        }
    },
    "Parameters": {
        "C1": "SDSS_g_Absolute",
        "lookat_x": "150",
        "lookat_y": "50",
        "lookat_z": "50",
        "sky_x": "0.2090429365634918",
        "sky_y": "0.559632420539856",
        "sky_z": "0.8019428849220276",
        ... // Additional parameters removed for brevity.
    }
}
```

FIGURE 5.3: A JSON representation of application state, which can be saved at runtime and reloaded to restore a previous interactive visualisation session.

Requirement **R.8** is addressed by allowing the user to capture an image of the current visualisation session, which can be named and downloaded as a JPEG file via HTTP.

Once satisfied with the results of an interactive filtering process using the features described in Section 5.3.3.3, the user can opt to store the filtered dataset to a specified location on the server, addressing requirement **R.9**. When embedded in a web service, this extends to a direct download of the dataset via HTTP. In order to support this, the generic file interface described in Section 5.3.3.2 was extended with an additional function:

write_filtered Write a filtered dataset based on the specified filter map.

Finally, to address requirement **R.10**, the saveable file containing a JSON representation of the server state was extended to contain a list of currently currently applied filters, and reformatted to be simply human readable. Figure 5.3 shows an example of such a state file for a small sample dataset, with many of the additional key-value parameters removed for brevity. This meta-data state file is also provided with image and data downloads as a record of the source data and actions that occurred to generate the output files, and providing a means to regenerate any visualisation output as necessary.

5.3.3.5 User Interface

Meeting the requirements of Section 5.3.3 necessitates a user interface. As a tool for embedding into existing web portals, it is important to support these portals in a general way; ideally, in a production environment, an embedded tool should employ an interface design that matches the look and feel of the rest of the portal. As such, it is important not to enforce a fixed user interface design.

The current interface, utilised in the following section (5.4), is based on the built-in dynamic interface generator detailed in Section 4.2.4.5. This generator by default exploits datGUI, a lightweight graphical interface library successfully used in other web-based visualisation tools such as Vohl, Barnes, et al. (2016). This approach provides a quick and simple means of modifying and extending the user interface with override-able extension points. On integration with a production web portal, it is expected that each of the interface options will be overriden using the framework and design of choice for the specific portal.

5.4 Visualisation in Practice

Starting from existing scenarios utilizing data from the TAO database, the following subsections replicate some of the early steps taken in the scientific process to demonstrate the utility of an interactive 3D web visualization tool meeting the requirements outlined in this context. The two use cases introduced in Section 5.3.1 are recreated by applying Splotch to TAO data, demonstrating the capability to explore and interact with galaxy catalogues, build up advanced filters for target selections, and to subset and extract a new galaxy catalogue by generating derived data through field combinations.

Framerates reported in this section are recorded with the Splotch visualization server running on Swan, a Cray XC50 located at the Cray computing facility in Wisconsin, USA. Each node consists of 2 Broadwell 22-core Xeon CPUs clocked at 2.2Ghz. The web client is running on a Macbook Pro (early 2013 model) with 2.7 GHz Intel Core i7, and Mozilla Firefox 57.0.4, at the University of Portsmouth in the UK.

5.4.1 Exploring the environments of recently merged galaxies

As detailed in Section 5.3.1, for this use case investigating the properties of galaxies in cosmic voids is a key science goal. The initial dataset is a 250 h^{-1} Mpc x 250 h^{-1} Mpc x 20 h^{-1} Mpc subset of the galaxy catalog. In many contexts a random selection here is fine, however it is also possible to tune the location of the sample to increase the number of voids and therefore targets for studying. Visualization can allow the user to undertake a visual process of sampling to identify relevant sections of the box, in order to increase the potential number of useful galaxies.

To illustrate this, a selection of galaxies is extracted starting with the full SAGE dataset (500 h^{-1} Mpc^3) at *z=0*, extracting 3D galaxy positions, *Total stellar mass* and *Time of last major merger*. This consists of 15619131 galaxies.

Using 2 compute nodes of the test system Swan, it is possible to visualize the full volume interactively at >10 fps, and begin to visually identify locations to filter remotely from the web browser of a local laptop, demonstrated in Figure 5.4. Subsequently, Figures 5.5, 5.6, 5.7, and 5.8, document the incremental application of filters to match the initial selection criteria of Penny et al. (2015).

In this use case, the initial selection becomes an interactive process supporting visual input and feedback, with the ability to undo-redo filtering and in real-time explore the environment and history of galaxy mergers before downloading specific subsets for further analysis.

5.4.2 Target Selection in the Large Scale Galaxy Distribution

As introduced in Section 5.3.1, finding a specific galaxy population amongst a wider distribution of galaxies can require multiple steps of data analysis and filtering. A visual approach can allow users to analyse, explore, and filter their data, even generating new data, before leaving the web portal. An example of this is demonstrated here, focusing on the investigation of superclusters in a wider galaxy distribution.

Figure 5.9 exhibits a 20 deg^2 lightcone where $0 < z < 0.1$. Built using the lightcone science module on top of the Millennium simulation and SAGE galaxy model, TAO calculates the spectral energy distribution for SDSS bandpass filters using the model of Conroy, Gunn, and White (2009). Galaxies are coloured relative to SDSS absolute *g* filter, while size and intensity are further scaled by total stellar mass for visual emphasis. Using a single compute node of the test system Swan, it is possible to visualize the full volume of 136298 galaxies interactively at >30 FPS.

To investigate the rare but significant superclusters in the galaxy distribution, a filter is created that extracts only galaxies where mVir > $5x10^{13}$ $h^{-1}M_{\odot}$, shown in Figure 5.10. Following this, Figure 5.11 demonstrates the use of a combinatory filter, colour-coding galaxies by the difference between SDSS bands *g* and *r*. This further highlights the age distribution of galaxies within such massive structures, where bluer galaxies (which tend to be younger and have higher star formation rates) live preferentially on the outskirts, while the redder galaxies (and hence older and mostly devoid of star formation) occupy the cluster cores. The series of figures demonstrate an interactive and in-situ process where the user can find unique populations, such as galaxy clusters and the distribution of galaxies within such structures, which can then be imaged without the need for external analysis. The subsequent output, images and data, can then be jointly downloaded to local storage for subsequent use and publication.

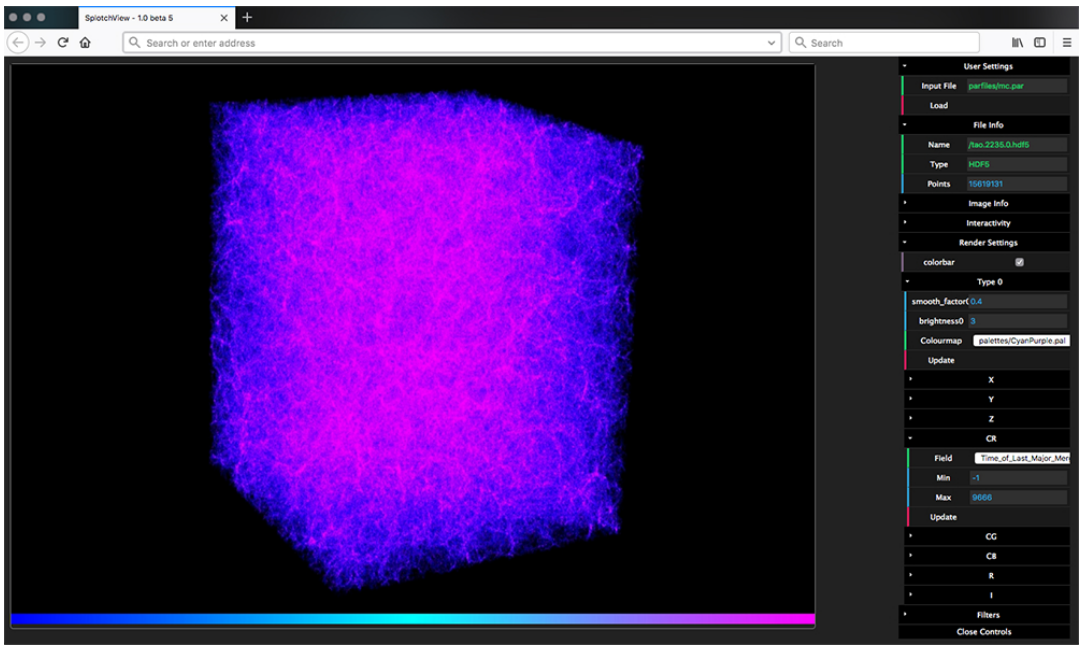


FIGURE 5.4: Splotch: the full SAGE dataset at *z=0*, 500 h^{-1} Mpc x 500 h^{-1} Mpc x 500 h^{-1} Mpc, coloured by *Time of last major merger*. The server is running on a HPC cluster, while the client is on a local laptop. Mouse/keyboard controls allow the user to rotate and zoom, while the interface to the right allows to change colour palettes, modify visualized quantities and more.

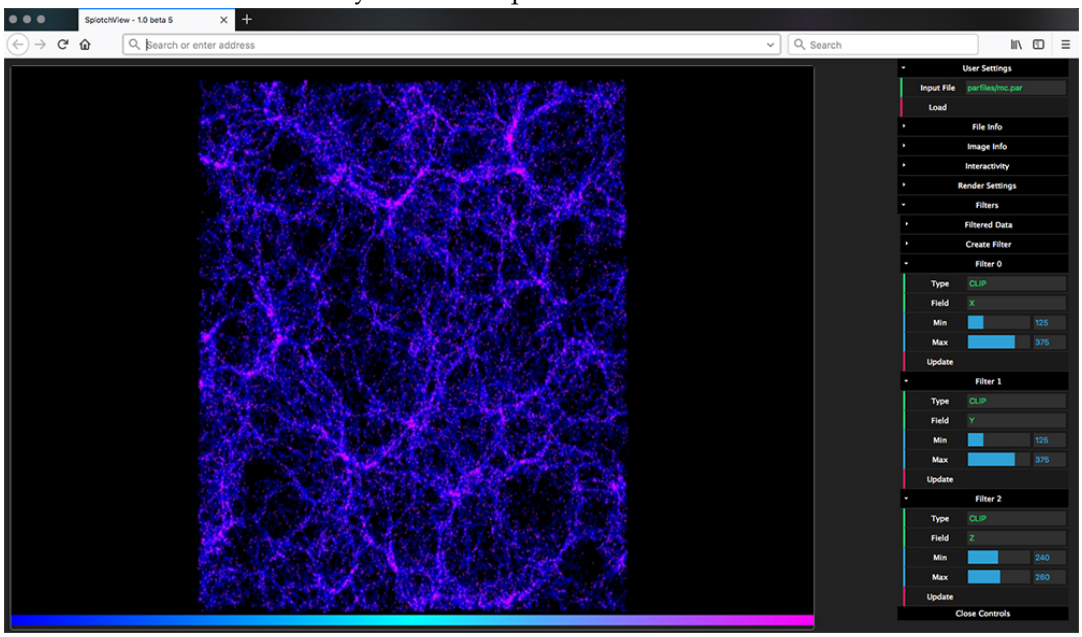


FIGURE 5.5: Splotch: clipping filters applied to the data of Figure 5.4 extract a 250 h^{-1} Mpc x 250 h^{-1} Mpc x 20 h^{-1} Mpc sub-slice. The user zooms in to fill the screen with the filtered dataset. The menu for applying and modifying filters can be seen in the lower right of the figure.

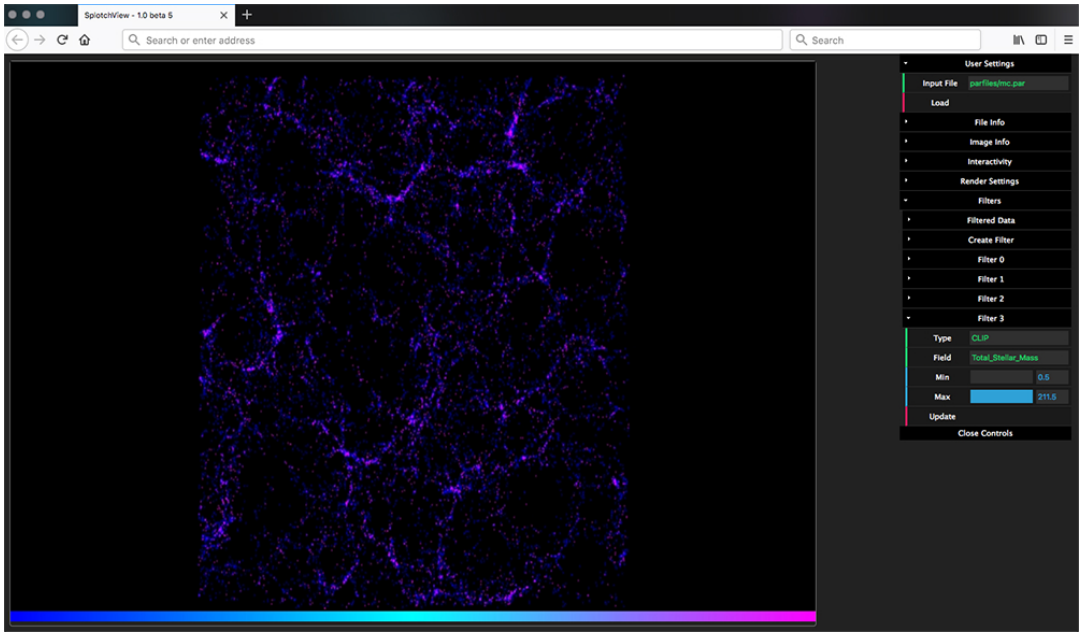


FIGURE 5.6: Splotch: further clipping filters applied to the data of Figure 5.5 extract a sub-slice containing only galaxies with stellar mass > $5x10^9$ $h^{-1}M_{\odot}$.

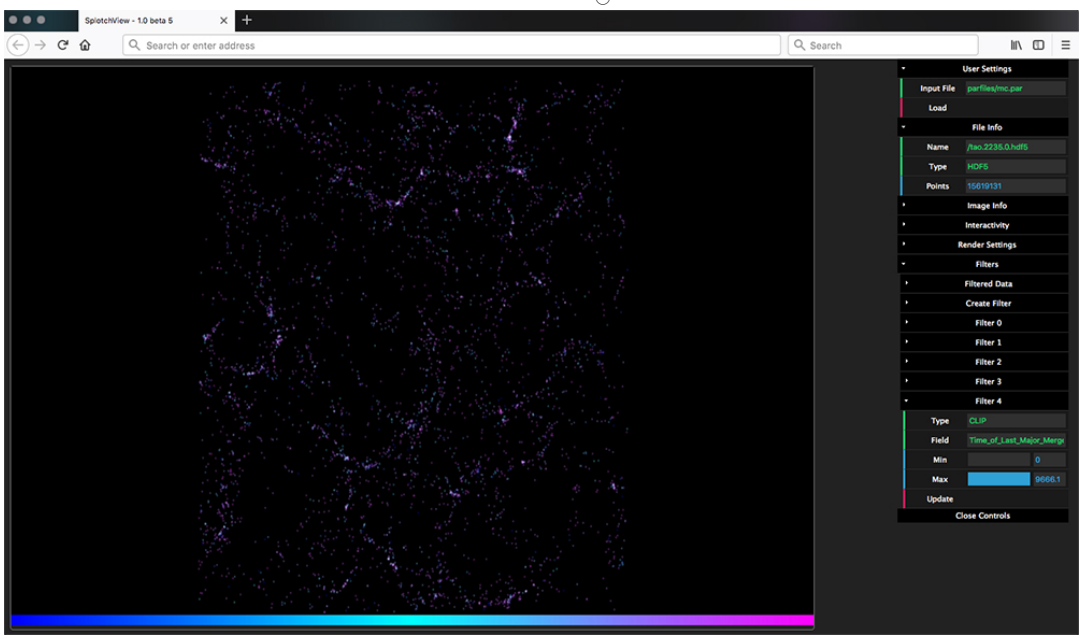


FIGURE 5.7: Splotch: Extending the filtering process of Figure 5.6, an additional filter is placed on *Time since last major merger* showing only galaxies who have undergone a major merger at some point in their evolutionary history.

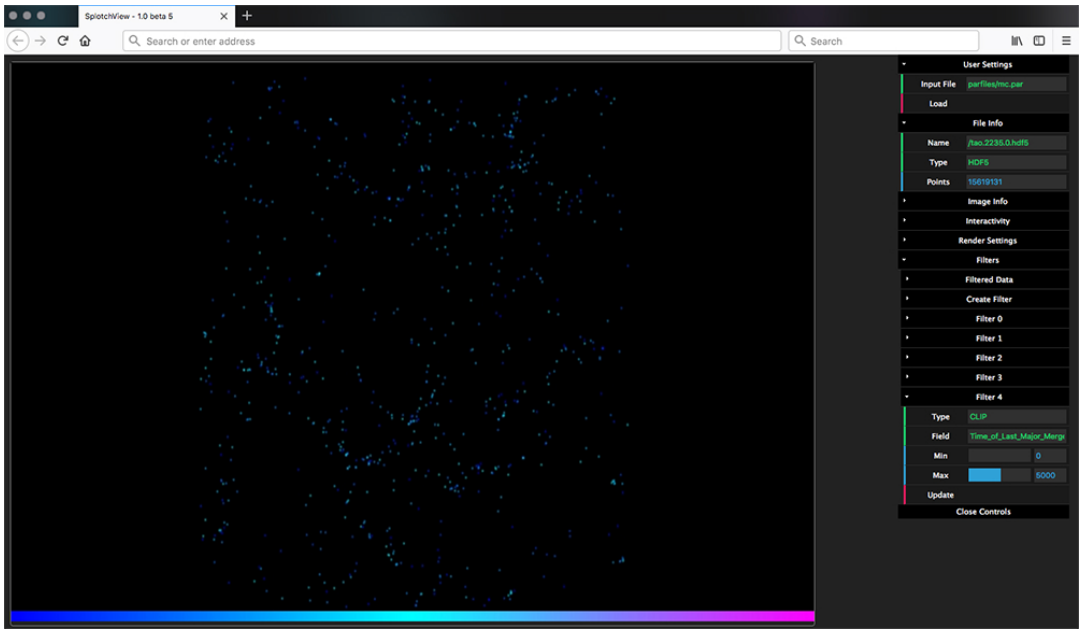


FIGURE 5.8: Splotch: Exploring the merger history of galaxies, the filter on *Time since last major merger* is modified to only include galaxies with a merger in the last 5 Gyr. The filter menu allows this dataset to be saved locally, along with visualization images and interaction history.

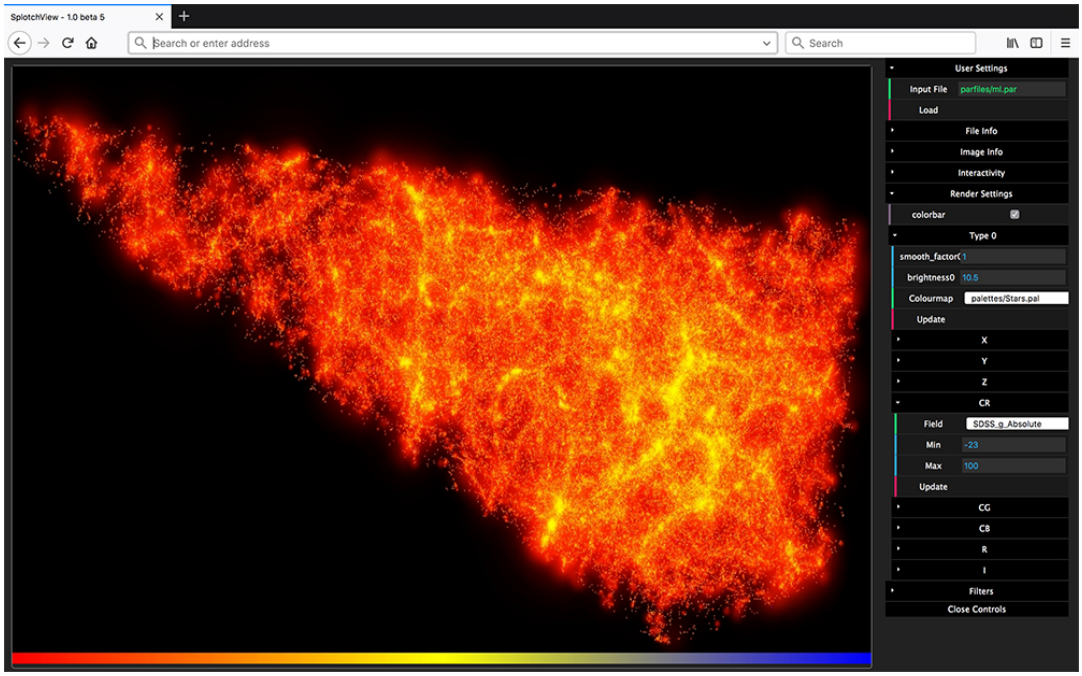


FIGURE 5.9: Splotch: A 20 deg^2 lightcone where $0 < z < 0.1$. Built using the TAO lightcone science module on top of the Millennium with spectral energy distribution computed for SDSS bandpass filters using the model of Conroy, Gunn, and White (2009). Galaxies are colouring according to SDSS Absolute G field, while size and intensity are scaled by total stellar mass for visual emphasis.

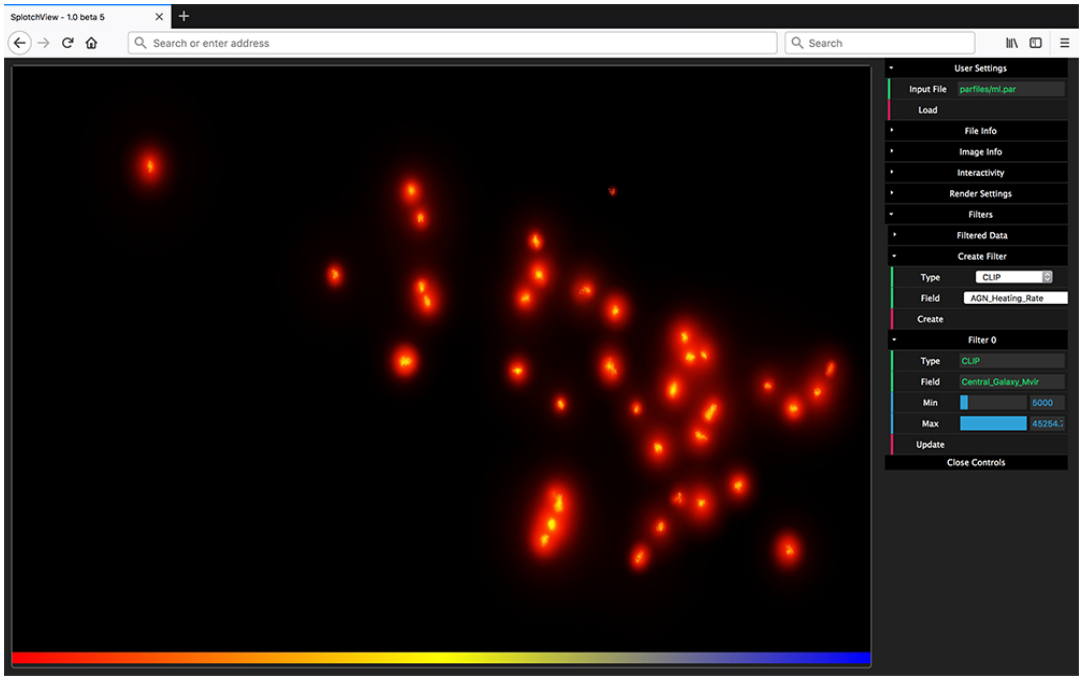


FIGURE 5.10: Splotch: the lightcone of Figure 5.9 is filtered such that only galaxies with central galaxy mVir > $5x10^{13}$ remain.

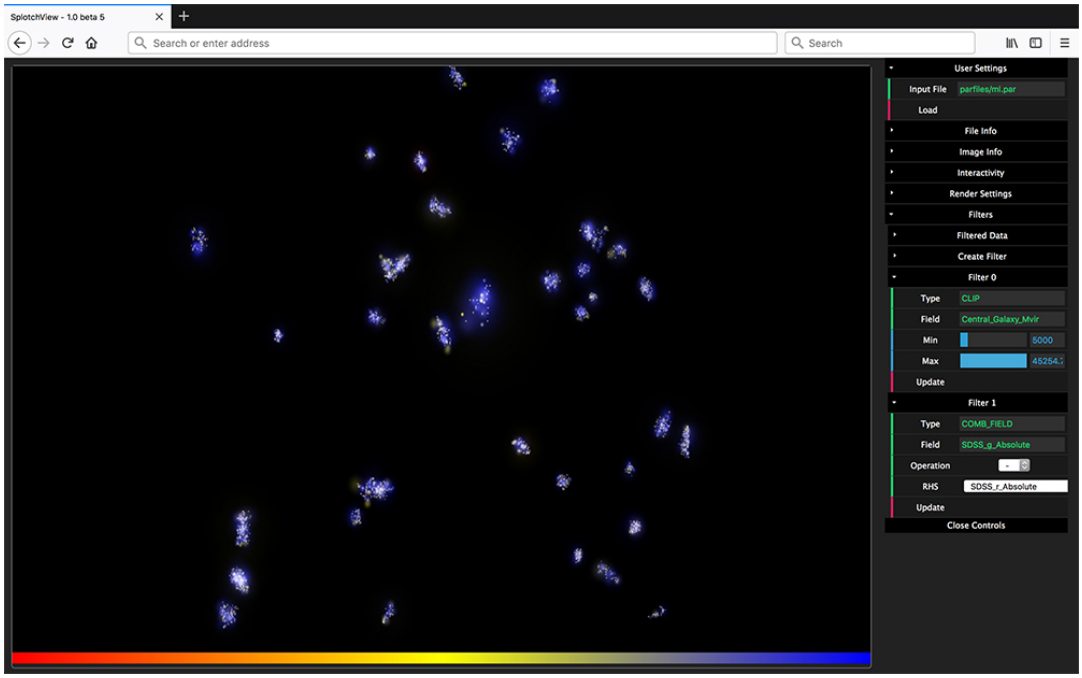


FIGURE 5.11: Splotch: a combinatory filter is applied to the remaining galaxies of Figure 5.10. The visualized field represents (G-R), while the view has been rotated to align with the direction of increasing redshift.

5.5 Discussion

Interactive visualization can provide an added value to the process of extracting theoretical datasets from newly emerging web portals. As demonstrated in Section 5.4, some of the common scenarios in which a scientist may require data from a theoretical web portal can be accomplished in a more fluid manner with visualization and interactive data filtering techniques. This section of the thesis has demonstrated a client-server tool for interactive web visualization, which can be seen as an embeddable module within a more general web framework. This work is applicable in multiple facilities, with the initial target being TAO. However, there are further challenges to be addressed before the system is integrated in a production facility, such as the integration of interactive jobs to the scheduling system of TAO. As discussed previously, the traditional model of running jobs on HPC systems is batch. For interactive visualization to work effectively in this model, the user and scheduling system must be able to cooperate to schedule the resources at a time that is acceptable. However, it is becoming more common for high performance systems to have a variety of queues for varying job size, hardware, and memory requirements. It is starting to become common to include interactive queue with dedicated resources, precisely for jobs such as visualization, debugging and other interactive tasks. In this model it is feasible to imagine the ideal scenario where the user requests an interactive session and receives it instantly.

Furthermore, it must be considered that users of the TAO service are likely not also users of the underlying HPC service, and as such they should not be able to exploit the rendering server in order to access restricted data. In consideration of this, Splotch contains a compile-time option to restrict the ability to load files direct from a file path on a per-user basis. Instead, the user may only loading datasets from a specific list which can be provided directly by the web framework.

As discussed in Section 5.2.3, there are a few existing tools that could enable the type of usage discussed here. Paraview Web is the most advanced tool for connecting HPC and web visualization, and has already been proven useful within the context of PDACS (see Section 5.2.1). A key difference in Splotch is the optimized support for high quality rendering of particle data using large and heterogeneous supercomputing systems. Splotch is envisioned as a tool that can be used entirely in its own right as exemplified here, or alternatively embedded within a larger and more generic visualization package. For example, Splotch is already exploited as a batch rendering mode within VisIVO, and there has been some work toward building a Splotch plugin for Paraview. The aim for this work is that the remote and interactive Splotch is considered a lightweight alternative to more general visualization software packages, ideal for building into a web module of an online virtual observatory.

The work presented shows the viability and efficacy of remote visualization as a support for theoretical data web portals, and the functionalities for data processing

can already be useful for scientific applications (Section 5.4). The next steps are full integration in a production service, along with further investigation of advanced visualization methods in this context towards a future vision of a user interactively and in real-time making virtual observations as a part of their web-based analysis workflow.

In relation to this, a further step to be taken is regarding comparative visualization. Along with quantitative methods, comparative approaches to visualization have been used to good effect for tasks such as verifying scientific simulations (Ahrens, Heitmann, et al., 2010) and analysing large sets of multi-dimensional data (Vohl, Barnes, et al., 2016). Whilst 3D visualization is inherently suited to viewing of theoretical astronomical data, many astronomers, particularly those from observational backgrounds, are more accustomed to viewing virtual observations. These provide one of the crucial links between theoretical and observational astronomy, and a common factor amongst each of the portals reviewed in Section 5.2.1 is the inclusion of virtual observations (as introduced in Chapter 2).

An experimental science module currently available in TAO is the mock imaging module, which employs the SkyMaker software package (Bertin, 2009) to generate mock telescope images considering common observational phenomena such as aperture, optical defects, and others not commonly used in 3D visualization. These images facilitate understanding of the relationship between properties of galaxies and real observations, and can be used to compare synthetic datasets from different sources with each other and telescope observations.

Mock imaging is traditionally not an interactive process. The user sets initial parameters and provides input data, then launches a job to generate an image. In future, it would be ideal to tie this process to the 3D visualization module, such that a user can interactively filter their data in 3D, then launch an on-the-fly mock image generator to see the effects of their interaction in a more recognizable format. While mock image generation need not be as interactive as 3D visualization, the viability of this approach depends on a fast mock imaging tool that can generate images in a reasonable time frame. As such, future work includes the evaluation of mock imaging tools and their suitability for this use case.

5.6 Summary

This chapter addresses the objective **O.4**: *'Discover if and how high performance visualisation can support astronomers in typical web-based access and analysis scenarios'*.

A novel survey of visualisation capabilities in existing web observatories for theoretical astronomical data is presented, identifying a clear lack of interactive 3D visualisation (Section 5.2.1). A general process for accessing theory data in web portals is described, and an improved process is presented based on interactive 3D visualisation (Section 5.2.2). A set of general requirements are identified for interactive visual discovery in theoretical astronomy web portals through analysis of typical use cases,

and a demonstrative tool is developed based on these requirements and the work of previous chapters (Section 5.3). The utility of this approach is then demonstrated though reconstruction of the analysed use cases (Section 5.4).

In view of the question posed: **Q.4**: *'How can remote high performance visualisation facilitate access and analysis of large astronomical datasets on the web?',* this chapter has demonstrated an approach for access and analysis of large astronomical datasets on the web and illustrated how this approach can facilitate astronomers in accessing and analysing data in the context of theoretical web portals.

Chapter 6

Summary and Future Directions

6.1 Research Summary and Contributions

Data-intensive scientific computing has become a prominent paradigm in the quest to answer some of the most important questions of our time. Experimentation, observation, and simulation are key tools in the scientific toolbox used to try and answer such questions. As discussed in Chapter 1, the increasing size and complexity of the environments surrounding such tools poses significant challenges for modern scientists. Processing, moving, storing, sharing, and accessing big scientific data is becoming an increasingly difficult set of tasks requiring researchers with diverse range of technical skills. The inexorable march of technological progress, through increasingly large and complex instrumentation and computing systems, is ensuring that such tasks will only become more difficult in the future. Data production is increasing at a phenomenal rate, and the analysis tools in the arsenal of the modern day scientist must evolve in equal measures to cope with current and upcoming data challenges. It is thus essential that those focusing on technological innovation work to do this in conjunction with the scientists facing these issues, bringing together expertise across disciplines to address technological challenges that may inhibit scientific progress. This thesis poses four key research questions with a view to, in part, find a solution to these challenges.

In the course of answering these research questions, the work of this thesis has contributed in a variety of ways to knowledge in the fields of high performance visualisation and astronomy. In Chapter 2, an improved optical model for the high performance visualisation algorithm Splotch was conceived and implemented based on an understanding of the science of radiative transport underpinning volume rendering (2.3). Building on this improved model, a new approach was proposed for modelling and visualising galaxies with dust lanes based on observed data, simulated data, and domain expertise; the approach was then demonstrated in the context of the well-known spiral disk galaxy M83 (2.4). Principles drawn from computer graphics were exploited to propose and demonstrate a new approach for simple modelling of direct lighting in particle based astrophysical simulations (2.5). Based on the domain expertise gained from this work, finally knowledge transfer was proposed back to computer graphics relating to coupled lighting and pre-computed

opacity maps (2.5).

In Chapter 3, first a series of trends in emerging architectures for high performance computing were identified (3.2). Then an optimisation approach was demonstrated targeting these trends to achieve performance portability for Splotch on an emerging architecture (3.3). As part of this work, GPU-based atomic operations were evaluated for updating images in volume splatting and found superior in performance to manual synchronisation techniques (3.4). The performance of volume splatting on three high performance computing architectures was compared, demonstrating the feasibility of achieving performance portability across these architectures and illustrating the performance variability for specific kernel types (3.5). Building on the improved optical model of Chapter 2, a new approach for volume splatting particles in distributed memory systems was proposed and demonstrated based on a KD-tree, domain boundary, and image compositing solutions (3.6). Finally using the experiences gained throughout this chapter, and based on the identified trends, two methodologies for algorithm optimisation were developed: firstly for optimisation of general algorithms on future architectures, and secondly for optimisation of volume splatting algorithms on future architectures (3.7).

Chapter 4 opened with the development of a novel classification scheme to describe the ways in which remote applications can run on high performance computing systems (4.2.2). A review of web-enabled real-time interactive remote applications was then presented in the context of this classification scheme, where a clear lack of general approach for building Direct Remote Web (DRW) or Indirect Remote Web (IRW) based remote applications was identified (4.2.3). To address this, an open source framework was built implementing a general approach for building DRW and IRW based remote applications (4.2.4). The framework was then demonstrated via the transformation of a batch-executable HPC visualisation application into a web-based interactive application (4.3). The performance of this new application was evaluated across three HPC architectures, demonstrating the feasibility of real time interaction and parallel scalability (4.3.3).

In Chapter 5 a novel survey of existing web observatories for theoretical astronomical data was presented in the context of visualisation, where a lack of interactive 3D visualisation was identified (5.2.1). To address this lack, first a general process for accessing theory data in a web portal was described, leading to an improved process based on the use of interactive visualisation (5.2.2). Then a series of exemplary use cases for a specific web portal were analysed to identify a general set of requirements for a new visualisation approach within such web portals (5.3.1, 5.3.2). A demonstrative tool was then developed based on these requirements and the work of previous chapters (5.3.3). Finally, the utility of such an approach was demonstrated though reconstruction of the analysed use cases (5.4).

6.2 Future Directions

In addressing the research questions as outlined in Section 1.3, this work has opened up a variety of avenues for future work. This section highlights five of these future directions.

6.2.1 High Quality Illumination models for Particle Based Astronomical Visualisation

Chapter 2 introduced an extended optical model in Splotch, aimed at supporting improved physical motivations for particle visualisation, with a supporting parallel algorithm presented in Chapter 3. As highlighted in Section 2.5.5, there are further steps that can be taken to increase the physical realism during rendering. The future vision for this work in terms of rendering quality is to explore coupling with the STARRAD algorithm to introduce shadows for visualising astronomical simulations with few significant light sources, and further investigation into the potential for more advanced lighting approaches. This should be coupled with further investigation into physically motivated transfer functions, such as the inclusion of robust absorption maps in relevant cases (the Rosseland mean opacity maps used in Section 2.5.3, for example), the use of cosmological simulation data to inform particle distributions, and a production integration of the parallel rendering algorithm. The work of Chapters 4 and 5 provide a framework within which these further investigations can be performed, introducing the potential for advanced visualisation widgets such as interactive transfer function editors that can help explore and tune new visualisation techniques. In the longer term, this should be combined with a wider investigation into the potential for coupling astrophysical radiative transport approaches with computer graphics, with particular consideration to the generation of physically realistic images for quantitative comparisons (as discussed in Section 6.2.4).

6.2.2 Interactive Galaxy Modelling

Chapter 2 introduced a novel galaxy modelling and visualisation pipeline, based on a wide range of observed galaxy imaging data and the improved optical model in Splotch. As discussed in Section 2.4.6, there are several promising directions for introducing further physically motivated parameters to the modelling and visualisation pipeline (i.e. Table 2.3). Beyond this, the remote, interactive, and web capabilities introduced in Splotch in Chapters 3 to 5 provide the foundations to further develop this pipeline into a fully interactive application, where the user can modify both modelling and visualisation parameters on the fly. Such an application would allow the user to effectively tune the galaxy modelling process, interactively trialling new combinations of parameters with immediate visual feedback and intuitive exploration of the results. Such interactive modelling applications have already seen

some success in astronomy (e.g. Shape (Steffen et al., 2011)), and the future vision of the Splotch modelling pipeline is an interactive tool to build complex and accurate models of galaxy neighbourhoods from the comfort of a web browser, creating fly-through animations and interactive experiences for research, education, and outreach.

6.2.3 Performance Portable Visualisation

Chapter 3 addresses the growing trend of heterogeneous hardware in high performance computing systems, with porting and optimisation experiences for emerging and evolving many-core accelerators. Section 3.8 discussed the growing interest in the approaches to performance portability, and a promising future line of research revolves around a study of performance portable visualisation. Key topics would build on the work of Chapter 3 for many-core and GPU hardware, extending the hardware focus to more recently introduced FPGAs for high performance computing (e.g. the recent Intel Stratix 10). The rising complexity of memory hierarchies in high performance computing systems, with the introduction of high bandwidth and non-volatile memories, must also be taken into consideration, for example through high level memory abstractions. These should be combined with a thorough investigation of the variety of new and evolved programming models for heterogeneous hardware, for example revisiting OpenCL, which has grown in popularity for tasks such as FPGA programming. The future vision of this work is a visualisation tool that can achieve good performance on any of the typical machines one may available to use in a high performance computing centre, through a low reliance on external libraries and a strong focus on performance portable software approaches.

6.2.4 Web Integrated Visualisation for Astronomy

Chapter 4 introduces a web framework for visualisation supported by high performance computing resources, based around Splotch and the WSRTI library. The utility of such a framework is demonstrated in the context of theoretical virtual observatories in Chapter 5, via a prototype tool in the context of the Theoretical Astrophysical Observatory. A future direction based on this work, as discussed in Section 5.5, would be a production level integration to TAO; the first main focus of this work would be addressing the logistics of integrating with high performance infrastructure i.e. network restrictions, workload management integration, queue management, which could be completed in the context of the WSRTI library. The second main focus of the work would be further integration of realistic rendering to the astronomers workflow, building on the work of Chapters 2 and 5 to support e.g. interactive generation of virtual observations.

6.2.5 Scientific Visualisation Beyond Astronomy

The scientific context of this thesis has been observed and simulated point-like data from the astronomical domain. However, scientific visualisation is a multi-disciplinary field, with a long history of applications in many different fields, as introduced in Chapter 1. The data structures and volume rendering approaches described in Chapter 2 (Section 2.2) are ubiquitous across computational fields, as is the introduction of heterogeneous high performance computing systems and web-enabled research environments (for example, the Galaxy workflow engine for life sciences research as mentioned in Chapter 5). As such, there is potential for the work here to apply in other fields where particle methods are frequently used. As an example, an important problem in the Earth and environmental sciences is the tracking of volcanic ash clouds, which can be a danger for flights whilst airborne, and when settling poses further hazards to humans, infrastructure, and wildlife. Simulation methods for volcanic ash transport and dispersion are an important means of predicting potential hazard zones and, for instance, popular simulation codes such as PUFF (Searcy, Dean, and Stringer, 1998) (for volcanic ash) and NAME (Jones, Thomson, et al., 2007) (for atmospheric modelling) are based on Lagrangian particle methods, with similar data structures to SPH-based astronomy simulations. Other examples with potential for knowledge transfer include common computational fluid dynamics applications, such as aerospace or chemical engineering. The cross-domain application of remote, interactive, and web based particle visualisation methods as presented in this thesis are a promising future direction.

Appendix A

Splotch

A.1 Splotch Solution to the Radiative Transfer Equation

Here the Splotch per-particle solution to the radiative transfer equation is justified. Starting point, radiative transfer equation (Section 2.3.2):

$$\frac{dI(x)}{dx} = (E_p - A_p I(x))\rho_p(x) \tag{A.1}$$

Define:

$$Q_p = \frac{E_p}{A_p} \tag{A.2}$$

Multiply by $\frac{1}{A_p}$:

$$\frac{1}{A_p}\frac{dI(x)}{dx} = (Q_p - I(x))\rho_p(x) \tag{A.3}$$

Rearrange:

$$\frac{dI(x)}{(Q_p - I(x))} = A_p\rho_p(x)dx \tag{A.4}$$

Integration:

$$\int_{I_1}^{I_2} \frac{dI(x)}{(Q_p - I(x))} = \int_{x_1}^{x_2} A_p\rho_p(x)dx \tag{A.5}$$

Define:

$$Y = -\ln(Q - I(x)) \tag{A.6}$$

$$Y_2 - Y_1 = A_p \int_{x_1}^{x_2} \rho_p(x)dx \tag{A.7}$$

$$e^{Y_2} = e^{Y_1 + A_p \int_{x_1}^{x_2} \rho_p(x)dx} \tag{A.8}$$

$$e^{Y_2} = e^{Y_1} \cdot e^{A_p \int_{x_1}^{x_2} \rho_p(x)dx} \tag{A.9}$$

Expand Y:

$$\frac{1}{(Q_p - I_2(x)} = \frac{1}{(Q_p - I_1(x)} \cdot e^{A_p \int_{x_1}^{x_2} \rho_p(x)dx} \tag{A.10}$$

Rearrange:

$$I_2(x) - Q_p = (I_1(x) - Q_p) e^{-A_p \int_{x_1}^{x_2} \rho_p(x)dx} \tag{A.11}$$

Result:

$$I_{after} = (I_{before} - E_p/A_p) \exp(-A_p \int_{-\infty}^{\infty} \rho_p(x)\, dx) + E_p/A_p \tag{A.12}$$

Appendix B

Ethical Considerations

The following pages contain the required ethics review documentation:

- Ethical Review Acceptance Letter
- UPR16 Checklist

Dr Steve C Hand
Head of School

University of Portsmouth

School of Creative Technologies

School of Creative Technologies
University of Portsmouth
Eldon Building
Winston Churchill Avenue
Portsmouth PO1 2DJ
United Kingdom

T: +44 (0)23 9284 5460
E: ct-enquiry@port.ac.uk
W: www.port.ac.uk/ct

Dear Tim,

You will receive a letter in due course with your ethical review number which you should keep for your records. The body of the letter is outlined for you below:

I am pleased to inform you that the CCI Faculty Ethics Committee, based on the information you have provided in your initial application and your additional responses to our questions, has given your application for the study entitled 'High Performance Scientific Visualisation' (application date 29/05/2015), a favourable opinion.

This opinion has been given for this study only, and any changes in the conditions of the study may require you to re-apply for ethical review.

Although the Committee has given a favourable opinion, the final responsibility for the ethical conduct of this work lies, as always, with the researcher(s).

Please note that the Committee reserves the right to re-review this application should any concerns be raised about it in the future.

Your ethical review number is FO: 06/15 - 0095.

If you have any questions about this, please let me know.

Vaughan Powell

(Chair, CCI FEthC)

FORM UPR16

Research Ethics Review Checklist

Please include this completed form as an appendix to your thesis (see the Research Degrees Operational Handbook for more information

UNIVERSITY OF PORTSMOUTH

Postgraduate Research Student (PGRS) Information				**Student ID:**	427864
PGRS Name:	Timothy Dykes				
Department:	School of Creative Technologies	**First Supervisor:**	Mel Krokos		
Start Date: (or progression date for Prof Doc students)	**01/10/2014**				
Study Mode and Route:	Part-time ☐ Full-time ☒	MPhil ☐ PhD ☐	MD ☐ Professional Doctorate ☐		

Title of Thesis:	Harnessing Emerging Supercomputers for Remote and Interactive Visual Discovery in Astronomy
Thesis Word Count: (excluding ancillary data)	44880

If you are unsure about any of the following, please contact the local representative on your Faculty Ethics Committee for advice. Please note that it is your responsibility to follow the University's Ethics Policy and any relevant University, academic or professional guidelines in the conduct of your study

Although the Ethics Committee may have given your study a favourable opinion, the final responsibility for the ethical conduct of this work lies with the researcher(s).

UKRIO Finished Research Checklist:
(If you would like to know more about the checklist, please see your Faculty or Departmental Ethics Committee rep or see the online version of the full checklist at: http://www.ukrio.org/what-we-do/code-of-practice-for-research/)

a) Have all of your research and findings been reported accurately, honestly and within a reasonable time frame?	YES ☒ NO ☐
b) Have all contributions to knowledge been acknowledged?	YES ☒ NO ☐
c) Have you complied with all agreements relating to intellectual property, publication and authorship?	YES ☒ NO ☐
d) Has your research data been retained in a secure and accessible form and will it remain so for the required duration?	YES ☒ NO ☐
e) Does your research comply with all legal, ethical, and contractual requirements?	YES ☒ NO ☐

Candidate Statement:

I have considered the ethical dimensions of the above named research project, and have successfully obtained the necessary ethical approval(s)

Ethical review number(s) from Faculty Ethics Committee (or from NRES/SCREC):	FO: 06/15 - 0095

If you have *not* submitted your work for ethical review, and/or you have answered 'No' to one or more of questions a) to e), please explain below why this is so:

Signed *(PGRS)*:		**Date:** 19/12/2018

Bibliography

Agrawal, Ankit and Alok Choudhary (2016). "Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science". English (US). In: *APL Materials* 4.5. ISSN: 2166-532X. DOI: 10.1063/1.4946894.

Ahern, Sean (2012). "The Path to Exascale". In: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Ed. by E. Wes Bethel, Hank Childs, and Charles Hansen. Florida: Chapman & Hall/CRC. Chap. 15, pp. 331–347.

Ahrens, J., K. Heitmann, et al. (2010). "Verifying Scientific Simulations via Comparative and Quantitative Visualization". In: *IEEE Computer Graphics and Applications* 30.6, pp. 16–28. ISSN: 0272-1716. DOI: 10.1109/MCG.2010.100.

Ahrens, J., Li-Ta Lo, et al. (2008). "Petascale visualization: Approaches and initial results". In: *2008 Workshop on Ultrascale Visualization*, pp. 24–28. DOI: 10.1109/ULTRAVIS.2008.5154060.

Ahrens, James, Berk Geveci, and Charles Law (2005). "36 - ParaView: An End-User Tool for Large-Data Visualization". In: *Visualization Handbook*. Ed. by Charles D. Hansen and Chris R. Johnson. Burlington: Butterworth-Heinemann, pp. 717–731. ISBN: 978-0-12-387582-2. DOI: https://doi.org/10.1016/B978-012387582-2/50038-1. URL: https://www.sciencedirect.com/science/article/pii/B9780123875822500381.

Amdahl, Gene M. (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: ACM, pp. 483–485. DOI: 10.1145/1465482.1465560. URL: http://doi.acm.org/10.1145/1465482.1465560.

Arens, S. and G. Domik (2010). "A Survey of Transfer Functions Suitable for Volume Rendering". In: *Proceedings of the 8th IEEE/EG International Conference on Volume Graphics*. VG'10. Norrköping, Sweden: Eurographics Association, pp. 77–83. ISBN: 978-3-905674-23-1. DOI: 10.2312/VG/VG10/077-083. URL: http://dx.doi.org/10.2312/VG/VG10/077-083.

Ayachit, Utkarsh et al. (2015). "ParaView Catalyst: Enabling In Situ Data Analysis and Visualization". In: *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ISAV2015. Austin, TX, USA: ACM, pp. 25–29. ISBN: 978-1-4503-4003-8. DOI: 10.1145/2828612.2828624. URL: http://doi.acm.org/10.1145/2828612.2828624.

Baghsorkhi, Sara S. et al. (2012). "Efficient Performance Evaluation of Memory Hierarchy for Highly Multithreaded Graphics Processors". In: *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '12. New Orleans, Louisiana, USA: ACM, pp. 23–34. ISBN: 978-1-4503-1160-1. DOI: 10.1145/2145816.2145820. URL: http://doi.acm.org/10.1145/2145816.2145820.

Balaprakash, P. et al. (2018). "Autotuning in High-Performance Computing Applications". In: *Proceedings of the IEEE* 106.11, pp. 2068–2083. ISSN: 0018-9219. DOI: 10.1109/JPROC.2018.2841200.

Bastholm, Eric et al. (2018). "Challenges of real-time processing in HPC environments: the ASKAP experience". In: vol. 10707, p. 14. DOI: 10.1117/12.2313137. URL: https://doi.org/10.1117/12.2313137.

Becciani, U. et al. (2010). "VisIVO: Integrated Tools and Services for Large-Scale Astrophysical Visualization". In: *Publications of the Astronomical Society of the Pacific* 122.887, p. 119. URL: http://stacks.iop.org/1538-3873/122/i=887/a=119.

Belinda, L. (2014). *Intel Xeon Phi: applications and solutions catalogue*.

Bernyk, M. et al. (2016). "The Theoretical Astrophysical Observatory: Cloud-based Mock Galaxy Catalogs". In: *The Astrophysical Journal Supplement Series* 223, 9, p. 9. DOI: 10.3847/0067-0049/223/1/9. arXiv: 1403.5270.

Bertin, E. (2009). "SkyMaker: astronomical image simulations made easy." In: Mem. Soc. Astron. Italiana 80, p. 422.

Bessell, Michael S. (2005). "Standard Photometric Systems". In: *Annual Review of Astronomy and Astrophysics* 43.1, pp. 293–336. DOI: 10.1146/annurev.astro.41.082801.100251. eprint: https://doi.org/10.1146/annurev.astro.41.082801.100251. URL: https://doi.org/10.1146/annurev.astro.41.082801.100251.

Bethel, E. Wes (2012). "Introduction". In: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Ed. by E. Wes Bethel, Hank Childs, and Charles Hansen. Florida: Chapman & Hall/CRC. Chap. 1, pp. 1–6.

Bethel, E. Wes, Hank Childs, and Charles Hansen (2012). *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. 1st. Chapman & Hall/CRC.

Beyer, Johanna, Markus Hadwiger, and Hanspeter Pfister (2015). "State-of-the-Art in GPU-Based Large-Scale Volume Visualization". In: *Comput. Graph. Forum* 34.8, pp. 13–37. ISSN: 0167-7055. DOI: 10.1111/cgf.12605. URL: https://doi.org/10.1111/cgf.12605.

Biffi, V. et al. (2012). "Observing simulated galaxy clusters with PHOX: a novel X-ray photon simulator". In: MNRAS 420, pp. 3545–3556. DOI: 10.1111/j.1365-2966.2011.20278.x. arXiv: 1112.0314.

Blaizot, J., B. Guiderdoni, et al. (2004). "GALICS- III. Properties of Lyman-break galaxies at a redshift of 3". In: MNRAS 352, pp. 571–588. DOI: 10.1111/j.1365-2966.2004.07947.x. eprint: astro-ph/0310071.

Blaizot, J., Y. Wadadekar, et al. (2005). "MoMaF: the Mock Map Facility". In: MNRAS 360, pp. 159–175. DOI: 10.1111/j.1365-2966.2005.09019.x. eprint: astro-ph/0309305.

Boch, T. and P. Fernique (2014). "Aladin Lite: Embed your Sky in the Browser". In: *Astronomical Data Analysis Software and Systems XXIII*. Ed. by N. Manset and P. Forshay. Vol. 485. Astronomical Society of the Pacific Conference Series, p. 277.

Borovska, P. and D. Ivanova (2014). "Code optimization and scaling of the astrophysics software Gadget on Intel Xeon Phi". In: *PRACE White Paper*, http://www.prace-ri.eu/evaluation-intel-mic.

Bottrell, Connor et al. (2017). "Galaxies in the Illustris simulation as seen by the Sloan Digital Sky Survey – I. Bulge+disc decompositions, methods and biases". In: *Monthly Notices of the Royal Astronomical Society* 467.1, pp. 1033–1066. ISSN: 0035-8711. DOI: 10.1093/mnras/stx017. eprint: http://oup.prod.sis.lan/mnras/article-pdf/467/1/1033/18756235/stx017.pdf. URL: https://doi.org/10.1093/mnras/stx017.

Brodlie, K. W. et al., eds. (1992). *Scientific Visualization: Techniques and Applications*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0-387-54565-4.

Brooke, J.M. et al. (2003). "Computational steering in realitygrid". English. In: *Proceedings of the UK e-Science All Hands Meeting 2003, 2-4 September, Nottingham, UK*. Ed. by S.J. Cox. EPSRC, pp. 885–1/4. ISBN: 1-904425-11-9.

Brown, David K. et al. (2015). "JMS: An Open Source Workflow Management System and Web-Based Cluster Front-End for High Performance Computing". In: *PLOS ONE* 10.8, e0134273. DOI: 10.1371/journal.pone.0134273.

Brownlee, Carson et al. (2012). "A Study of Ray Tracing Large-scale Scientific Data in Two Widely Used Parallel Visualization Applications". In: *Eurographics Symposium on Parallel Graphics and Visualization*. Ed. by Hank Childs, Torsten Kuhlen, and Fabio Marton. The Eurographics Association. ISBN: 978-3-905674-35-4. DOI: 10.2312/EGPGV/EGPGV12/051-060.

Brunner, Robert J. et al. (2002). "Handbook of Massive Data Sets". In: ed. by James Abello, Panos M. Pardalos, and Mauricio G. C. Resende. Norwell, MA, USA: Kluwer Academic Publishers. Chap. Massive Datasets in Astronomy, pp. 931–979. ISBN: 1-4020-0489-3. URL: http://dl.acm.org/citation.cfm?id=779232.779260.

Bryan, G. L. and M. L. Norman (1997). "A Hybrid AMR Application for Cosmology and Astrophysics". In: *ArXiv Astrophysics e-prints*. eprint: astro-ph/9710187.

Buchty, Rainer et al. (2011). "A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators". In: *Concurrency and Computation: Practice and Experience* 24.7, pp. 663–675. DOI: 10.1002/cpe.1904. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.1904. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.1904.

Cabral, Brian, Nancy Cam, and Jim Foran (1994). "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware". In: *Proceedings of the 1994 Symposium on Volume Visualization*. VVS '94. Tysons Corner, Virginia, USA: ACM, pp. 91–98. ISBN: 0-89791-741-3. DOI: 10.1145/197938.197972. URL: http://doi.acm.org/10.1145/197938.197972.

Canon, S. et al. (2010). "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud". In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science(CLOUDCOM)*. Vol. 00, pp. 159–168. DOI: 10.1109/CloudCom.2010.69. URL: doi.ieeecomputersociety.org/10.1109/CloudCom.2010.69.

Carretero, Jorge et al. (2017). "CosmoHub and SciPIC: Massive cosmological data analysis, distribution and generation using a Big Data platform". In: *PoS* EPS-HEP2017, p. 488. DOI: 10.22323/1.314.0488.

Carretero, J. et al. (2015). "An algorithm to build mock galaxy catalogues using MICE simulations". In: MNRAS 447, pp. 646–670. DOI: 10.1093/mnras/stu2402. arXiv: 1411.3286.

Cascaval, C. et al. (2010). "A taxonomy of accelerator architectures and their programming models". In: *IBM Journal of Research and Development* 54.5, 5:1–5:10. ISSN: 0018-8646. DOI: 10.1147/JRD.2010.2059721.

Cerezo, Eva et al. (2005). "A Survey on Participating Media Rendering Techniques". In: *Vis. Comput.* 21.5, pp. 303–328. ISSN: 0178-2789. DOI: 10.1007/s00371-005-0287-1. URL: http://dx.doi.org/10.1007/s00371-005-0287-1.

Chandrasekhar, Subrahmanyan (1950). *Radiative transfer*. Oxford University Press.

Chard, R. et al. (2014). "PDACS - A Portal for Data Analysis Services for Cosmological Simulations". In: *2014 9th Gateway Computing Environments Workshop*, pp. 30–33. DOI: 10.1109/GCE.2014.7.

Childs, Hank (2012). "Parallel Visualization Frameowrks". In: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Ed. by E. Wes Bethel, Hank Childs, and Charles Hansen. Florida: Chapman & Hall/CRC. Chap. 2, pp. 9–23.

Childs, Hank et al. (2011). "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data". In: *In Proceedings of SciDAC*.

Cholia, S., D. Skinner, and J. Boverhof (2010). "NEWT: A RESTful service for building High Performance Computing web applications". In: *2010 Gateway Computing Environments Workshop (GCE)*, pp. 1–11. DOI: 10.1109/GCE.2010.5676125.

Chrysos, G. (2012). "Intel Xeon Phi coprocessor (codename Knights Corner)". In: Intel. 24th Annual Hot Chips Symposium on High Performance Chips, San Francisco, USA.

Church, Philip, Andrzej Goscinski, and Christophe Lefèvre (2015). "Exposing HPC and sequential applications as services through the development and deployment of a SaaS cloud". In: *Future Generation Computer Systems* 43-44, pp. 24–37. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2014.10.001. URL: http://www.sciencedirect.com/science/article/pii/S0167739X1400185X.

Cito, J. and H. C. Gall (2016). "Using Docker Containers to Improve Reproducibility in Software Engineering Research". In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 906–907.

Colagrossi, Andrea and Maurizio Landrini (2003). "Numerical simulation of interfacial flows by smoothed particle hydrodynamics". In: *Journal of Computational Physics* 191.2, pp. 448–475. ISSN: 0021-9991. DOI: https://doi.org/10.1016/S0021-9991(03)00324-3. URL: http://www.sciencedirect.com/science/article/pii/S0021999103003243.

Colefax Research (2017). *Capabilities of Intel® AVX-512 in Intel® Xeon® Scalable Processors (Skylake)*. Tech. rep. URL: https://colfaxresearch.com/skl-avx512/ (visited on 08/12/2018).

Colwell, R. (2013). "The chip design game at the end of Moore's law". In: *2013 IEEE Hot Chips 25 Symposium (HCS)*, pp. 1–16. DOI: 10.1109/HOTCHIPS.2013.7478302.

Comparato, M. et al. (2007). "Visualization, Exploration, and Data Analysis of Complex Astrophysical Data". In: *Publications of the Astronomical Society of the Pacific* 119.858, p. 898. URL: http://stacks.iop.org/1538-3873/119/i=858/a=898.

Connolly, A. J. et al. (2014). "An end-to-end simulation framework for the Large Synoptic Survey Telescope". In: *Modeling, Systems Engineering, and Project Management for Astronomy VI*. Vol. 9150. Proc. SPIE, p. 915014. DOI: 10.1117/12.2054953.

Conroy, C., J. E. Gunn, and M. White (2009). "The Propagation of Uncertainties in Stellar Population Synthesis Modeling. I. The Relevance of Uncertain Aspects of Stellar Evolution and the Initial Mass Function to the Derived Physical Properties of Galaxies". In: ApJ 699, pp. 486–506. DOI: 10.1088/0004-637X/699/1/486. arXiv: 0809.4261.

Cossins, Peter J. (2010). "Smoothed Particle Hydrodynamics". PhD thesis. -.

Coulouris, George et al. (2011). *Distributed Systems: Concepts and Design*. 5th. USA: Addison-Wesley Publishing Company.

Courtland, R. (2015). *Gordon Moore: The Man Whose Name Means Progress*. URL: https://spectrum.ieee.org/computing/hardware/gordon-moore-the-man-whose-name-means-progress (visited on 07/30/2018).

Croton, D. J. et al. (2016). "Semi-Analytic Galaxy Evolution (SAGE): Model Calibration and Basic Results". In: ApJS 222, 22, p. 22. DOI: 10.3847/0067-0049/222/2/22. arXiv: 1601.04709.

Dale, D. A. et al. (2009). "The Spitzer Local Volume Legacy: Survey Description and Infrared Photometry". In: ApJ 703, pp. 517–556. DOI: 10.1088/0004-637X/703/1/517. arXiv: 0907.4722.

D'Alessio, Paola, Nuria Calvet, and Lee Hartmann (2001). "Accretion Disks around Young Objects. III. Grain Growth". In: ApJ 553, pp. 321–334. DOI: 10.1086/320655. arXiv: astro-ph/0101443 [astro-ph].

Damkroger, Trish (2017). *Unleashing High-Performance Computing Today and Tomorrow*. https://itpeernetwork.intel.com/unleashing-high-performance-computing/. (Visited on 11/13/2017).

Demchenko, Y. et al. (2013). "Addressing big data issues in Scientific Data Infrastructure". In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 48–55. DOI: 10.1109/CTS.2013.6567203.

Deslippe, J. et al. (2015). "Lessons Learned from Optimizing Science Kernels for Intel's "Knights Corner"' Architecture". In: *Computing in Science Engineering* 17.3, pp. 30–42. ISSN: 1521-9615. DOI: 10.1109/MCSE.2015.28.

Diaz, J., C. Muñoz-Caro, and A. Niño (2012). "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era". In: *IEEE Transactions on Parallel and Distributed Systems* 23.8, pp. 1369–1386. ISSN: 1045-9219. DOI: 10.1109/TPDS.2011.308.

Djorgovski, S. G. and R. Williams (2005). "Virtual Observatory: From Concept to Implementation". In: *From Clark Lake to the Long Wavelength Array: Bill Erickson's Radio Science*. Ed. by N. Kassim et al. Vol. 345. Astronomical Society of the Pacific Conference Series, p. 517. eprint: astro-ph/0504006.

Dolag, K., S. Borgani, et al. (2008). "Simulation Techniques for Cosmological Simulations". In: Space Sci. Rev. 134, pp. 229–268. DOI: 10.1007/s11214-008-9316-5. arXiv: 0801.1023 [astro-ph].

Dolag, K., M. Reinecke, et al. (2008). "Splotch: visualizing cosmological simulations". In: *New Journal of Physics* 10.12, 125006, p. 125006. DOI: 10.1088/1367-2630/10/12/125006. arXiv: 0807.1742.

Downing, Chris (2018). *How the Cloud Is Falling Short for HPC*. https://www.hpcwire.com/2018/03/15/how-the-cloud-is-falling-short-for-research-computing/.

Dreyer, J. L. E. (1888). "A New General Catalogue of Nebulæ and Clusters of Stars, being the Catalogue of the late Sir John F. W. Herschel, Bart, revised, corrected, and enlarged". In: MmRAS 49, p. 1.

Driver, S. P. et al. (2009). "GAMA: towards a physical understanding of galaxy formation". In: *Astronomy and Geophysics* 50.5, pp. 5.12–5.19. DOI: 10.1111/j.1468-4004.2009.50512.x. arXiv: 0910.5123 [astro-ph.CO].

Dubath, P. et al. (2017). "The Euclid Data Processing Challenges". In: *Astroinformatics*. Ed. by M. Brescia et al. Vol. 325. IAU Symposium, pp. 73–82. DOI: 10.1017/S1743921317001521. arXiv: 1701.08158 [astro-ph.IM].

Dykes, T., C. Gheller, M. Krokos, et al. (2015). "Accelerated 3D visualization of mock galaxy catalogues for the Dark Energy Survey". English. In: *Astronomical Data Analysis Software and Systems XXV*. Astronomical Society of the Pacific.

Dykes, T., C. Gheller, M. Rivi, et al. (2017). "Splotch: porting and optimizing for the Xeon Phi". In: *The International Journal of High Performance Computing Applications* 31.6, pp. 550–563. DOI: 10.1177/1094342016652713. eprint: https://doi.org/10.1177/1094342016652713. URL: https://doi.org/10.1177/1094342016652713.

Eeckhout, L. (2017). "Is Moore's Law Slowing Down? What's Next?" In: *IEEE Micro* 37.4, pp. 4–5. ISSN: 0272-1732. DOI: 10.1109/MM.2017.3211123.

Eisenstein, D. J. et al. (2001). "Spectroscopic Target Selection for the Sloan Digital Sky Survey: The Luminous Red Galaxy Sample". In: AJ 122, pp. 2267–2280. DOI: 10.1086/323717. eprint: astro-ph/0108153.

Elena, A. and I. Rungger (2014). "Enabling Smeagol on Xeon Phi: lessons learned". In: *PRACE White Paper*, http://www.prace-ri.eu/evaluation-intel-mic.

Elvins, T. Todd (1992). "A Survey of Algorithms for Volume Visualization". In: *SIGGRAPH Comput. Graph.* 26.3, pp. 194–201. ISSN: 0097-8930. DOI: 10.1145/142413.142427. URL: http://doi.acm.org/10.1145/142413.142427.

Euclid Consortium (2017). *Scientists of the Euclid Consortium announce the release of the largest simulated galaxy catalogue ever built.* Press Release. URL: http://www.euclid-ec.org/?page_id=4133.

Evans, Alun et al. (2014). "3D graphics on the web: A survey". In: *Computers & Graphics* 41, pp. 43–61.

Fang, J. et al. (2014). "Test-driving Intel Xeon Phi". In: *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*.

Faulkner, Andrew et al. (2010). "The Aperture Arrays for the SKA : the SKADS White Paper". In: SKA Memos.

Feigelson, Eric D. and G. Jogesh Babu (2012). "Big data in astronomy". In: *Significance* 9.4, pp. 22–25. DOI: 10.1111/j.1740-9713.2012.00587.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2012.00587.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1740-9713.2012.00587.x.

Ferguson, H. C. et al. (2009). "Astronomical Data Reduction and Analysis for the Next Decade". In: *astro2010: The Astronomy and Astrophysics Decadal Survey*. Vol. 2010. ArXiv Astrophysics e-prints.

Ferwerda, James A. (2003). "Three varieties of realism in computer graphics". In: vol. 5007, pp. 5007–8. DOI: 10.1117/12.473899. URL: https://doi.org/10.1117/12.473899.

Fette, I. and A. Melnikov (2011). *The WebSocket Protocol*. RFC 6455. http://www.rfc-editor.org/rfc/rfc6455.txt. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc6455.txt.

Fiore, S. et al. (2016). "Distributed and cloud-based multi-model analytics experiments on large volumes of climate change data in the earth system grid federation eco-system". In: *2016 IEEE International Conference on Big Data (Big Data)*, pp. 2911–2918. DOI: 10.1109/BigData.2016.7840941.

Fisher, David B. and Niv Drory (2011). "Demographics of Bulge Types within 11 Mpc and Implications for Galaxy Evolution". In: ApJ 733.2, L47, p. L47. DOI: 10.1088/2041-8205/733/2/L47. arXiv: 1104.0020 [astro-ph.CO].

Fletcher, C. A. (1988). *Computational Techniques for Fluid Dynamics*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0-387-18151-2.

Fluke, C. J. et al. (2011). "Astrophysical Supercomputing with GPUs: Critical Decisions for Early Adopters". In: PASA 28, pp. 15–27. DOI: 10.1071/AS10019. arXiv: 1008.4623 [astro-ph.IM].

Friendly, M. (2006). "A Brief History of Data Visualization". In: *Handbook of Computational Statistics: Data Visualization*. Ed. by C. Chen, W. Härdle, and A Unwin. Vol. III. (In press). Heidelberg: Springer-Verlag, ???–???

— (2009). "Milestones in the history of thematic cartography, statistical graphics, and data visualization". In: Web document. URL: http://datavis.ca/milestones/.

Gaburov, E. and Y. Cavecchi (2014). "Xeon Phi meets astrophysical fluid dynamics". In: *PRACE White Paper*, http://www.prace-ri.eu/evaluation-intel-mic.

Gadotti, D. A. (2009). "Structural properties of pseudo-bulges, classical bulges and elliptical galaxies: a Sloan Digital Sky Survey perspective". In: MNRAS 393, pp. 1531–1552. DOI: 10.1111/j.1365-2966.2008.14257.x. arXiv: 0810.1953.

Galvagni, M. et al. (2012). "The collapse of protoplanetary clumps formed through disc instability: 3D simulations of the pre-dissociation phase". In: *Monthly Notices of the Royal Astronomical Society* 427, pp. 1725–1740. DOI: 10.1111/j.1365-2966.2012.22096.x. arXiv: 1209.2129 [astro-ph.EP].

Gardner, Jonathan P. et al. (2006). "The James Webb Space Telescope". In: *Space Science Reviews* 123.4, pp. 485–606. ISSN: 1572-9672. DOI: 10.1007/s11214-006-8315-7. URL: https://doi.org/10.1007/s11214-006-8315-7.

Geer, D. (2005). "Chip makers turn to multicore processors". In: *Computer* 38.5, pp. 11–13. ISSN: 0018-9162. DOI: 10.1109/MC.2005.160.

Geist, Al and Daniel A Reed (2017). "A survey of high-performance computing scaling challenges". In: *The International Journal of High Performance Computing Applications* 31.1, pp. 104–113. DOI: 10.1177/1094342015597083. eprint: https://doi.org/10.1177/1094342015597083. URL: https://doi.org/10.1177/1094342015597083.

Gerlach, W. et al. (2014). "Skyport - Container-Based Execution Environment Management for Multi-cloud Scientific Workflows". In: *2014 5th International Workshop on Data-Intensive Computing in the Clouds*, pp. 25–32. DOI: 10.1109/DataCloud.2014.6.

Gesing, S. et al. (2015). "Science gateways - leveraging modeling and simulations in HPC infrastructures via increased usability". In: *2015 International Conference on High Performance Computing Simulation (HPCS)*, pp. 19–26. DOI: 10.1109/HPCSim.2015.7237017.

Gil de Paz, A. et al. (2007). "The GALEX Ultraviolet Atlas of Nearby Galaxies". In: ApJS 173, pp. 185–255. DOI: 10.1086/516636. eprint: astro-ph/0606440.

Giles, M. B. and I. Reguly (2014). "Trends in high-performance computing for engineering calculations". In: *Philosophical Transactions of the Royal Society of London Series A* 372, pp. 20130319–20130319. DOI: 10.1098/rsta.2013.0319.

Gingold, R. A. and J. J. Monaghan (1977). "Smoothed particle hydrodynamics - Theory and application to non-spherical stars". In: MNRAS 181, pp. 375–389. DOI: 10.1093/mnras/181.3.375.

Gloger, Wolfram (2006). *ptmalloc3*. http://www.malloc.de/en/.

Goecks, Jeremy et al. (2010). "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences". In: *Genome Biology* 11.8, R86. ISSN: 1474-760X. DOI: 10.1186/gb-2010-11-8-r86. URL: https://doi.org/10.1186/gb-2010-11-8-r86.

Goodale, Tom et al. (2003). "The Cactus Framework and Toolkit: Design and Applications". In: *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*. Berlin: Springer. URL: http://edoc.mpg.de/3341.

Goodman, A. A., M. A. Borkin, and T. P. Robitaille (2018). "New Thinking on, and with, Data Visualization". In: *ArXiv e-prints*. arXiv: 1805.11300 [astro-ph.IM].

Goodman, A. A., P. S. Udomprasert, et al. (2011). *Astronomy Visualization for Education and Outreach*. Boston, MA. URL: %5Curl%7Bhttp://aspbooks.org/custom/publications/paper/442-0659.html%7D.

Gorgolewski, Krzysztof J. et al. (2017). "BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods". In: *PLoS Computational Biology*.

Górski, K. M. et al. (2005). "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere". In: *The Astrophysical Journal* 622, pp. 759–771. DOI: 10.1086/427976. eprint: astro-ph/0409513.

Gupta, A. and D. Milojicic (2011). "Evaluation of HPC Applications on Cloud". In: *2011 Sixth Open Cirrus Summit*, pp. 22–26. DOI: 10.1109/OCS.2011.10.

Hassan, A. H., C. J. Fluke, et al. (2013). "Tera-scale astronomical data analysis and visualization". In: *Monthly Notices of the Royal Astronomical Society* 429.3, pp. 2442–2455. DOI: 10.1093/mnras/sts513. eprint: /oup/backfile/content_public/journal/mnras/429/3/10.1093/mnras/sts513/2/sts513.pdf. URL: http://dx.doi.org/10.1093/mnras/sts513.

Hassan, A. and C. J. Fluke (2011). "Scientific Visualization in Astronomy: Towards the Petascale Astronomy Era". In: PASA 28, pp. 150–170. DOI: 10.1071/AS10031. arXiv: 1102.5123 [astro-ph.IM].

Havran, V. (2000). "Heuristic Ray Shooting Algorithms". PhD thesis. Prague: Department of Computer Science and Engineering, Czech Technical University.

Hazra, R. (2013). "Driving industrial innovation on the path to exascale: from vision to reality". In: International Supercomputing Conference, Leipzig, Germany.

Heckbert, Paul S (1986). "Survey of Texture Mapping". In: *IEEE Comput. Graph. Appl.* 6.11, pp. 56–67. ISSN: 0272-1716. DOI: 10.1109/MCG.1986.276672. URL: http://dx.doi.org/10.1109/MCG.1986.276672.

Heinecke, A. et al. (2014). "Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

Helou, G. et al. (2004). "The Anatomy of Star Formation in NGC 300". In: ApJS 154, pp. 253–258. DOI: 10.1086/422640. eprint: astro-ph/0408248.

Hernquist, L. and N. Katz (1989). "TREESPH - A unification of SPH with the hierarchical tree method". In: ApJS 70, pp. 419–446. DOI: 10.1086/191344.

Hey, Tony, Stewart Tansley, and Kristin Tolle, eds. (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research. URL: http://research.microsoft.com/en-us/collaboration/fourthparadigm/.

Hildebrand, K., M. Magnor, and B. Fröhlich (2006). "3D reconstruction and visualization of spiral galaxies". In: *WSCG* 14, pp. 113–120. eprint: astro-ph/0111367.

Hoffman, Allan R. and Joseph F. Traub (1989). *Supercomputers: Directions in Technology and Applications*. Washington, DC: The National Academies Press. ISBN: 978-0-309-04088-4. DOI: 10.17226/1405. URL: https://www.nap.edu/catalog/1405/supercomputers-directions-in-technology-and-applications.

Hwang, Kai (1992). *Advanced Computer Architecture: Parallelism,Scalability,Programmability*. 1st. McGraw-Hill Higher Education. ISBN: 0070316228.

Hwu, W., K. Keutzer, and T. G. Mattson (2008). "The Concurrency Challenge". In: *IEEE Design Test of Computers* 25.4, pp. 312–320. ISSN: 0740-7475. DOI: 10.1109/MDT.2008.110.

Intel (2012a). *A guide to auto-vectorization with Intel C++ compilers*. Intel.

— (2012b). *How to use huge pages to improve application performance on Intel Xeon Phi coprocessor*. Intel.

— (2012c). *Intel Xeon Phi coprocessor instruction set architecture reference manual*. Intel.

— (2012d). *Optimization and performance tuning for Intel Xeon Phi coprocessors, Part 1: Optimization essentials*.

— (2012e). *Optimization and performance tuning for Intel Xeon Phi coprocessors, Part 2: Understanding and using hardware events*.

— (2012f). *Xeon Phi coprocessor system software developers guide*. Intel.

— (2013a). *Effective use of the Intel compiler's offload features*. Intel.

— (2013b). *Intel many-integrated-core architecture - advanced*. Intel.

Intel Corporation (2018). *Intel VTune Amplifier 2018 User's Guide*. URL: https://software.intel.com/en-us/vtune-amplifier-help (visited on 08/22/2018).

Isik-Ercan, Zeynep, Beomjin Kim, and Jeffrey Nowak (2010). "3D Visualization in Elementary Education Astronomy: Teaching Urban Second Graders about the Sun, Earth, and Moon". In: *Knowledge Management, Information Systems, E-Learning, and Sustainability Research*. Ed. by Miltiadis D. Lytras et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 500–505. ISBN: 978-3-642-16318-0.

Jagadish, H. V. et al. (2014). "Big Data and Its Technical Challenges". In: *Commun. ACM* 57.7, pp. 86–94. ISSN: 0001-0782. DOI: 10.1145/2611567. URL: http://doi.acm.org/10.1145/2611567.

Jeffers, J., J. Reinders, and A. Sodani (2016). *Intel Xeon Phi coprocessor high-performance programming: Knights Landing Edition*. 2nd ed. Morgan Kaufmann.

Jeffers, Jim (2016). *High Performance, High Fidelity, or Lower Cost Visualization, Pick Three!* IEEE Vis 2016. URL: http://sdvis.org/presentations/IEEEVis2016_SDVis.pdf.

Jin, Z. et al. (2010). "High-performance astrophysical visualization using Splotch". In: *ArXiv e-prints*. arXiv: 1004.1302 [astro-ph.IM].

Johnson, Christopher and Charles Hansen (2004). *Visualization Handbook*. Orlando, FL, USA: Academic Press, Inc. ISBN: 012387582X.

Jones, Andrew, David Thomson, et al. (2007). "The U.K. Met Office's Next-Generation Atmospheric Dispersion Model, NAME III". In: *Air Pollution Modeling and Its Application XVII*. Ed. by Carlos Borrego and Ann-Lise Norman. Boston, MA: Springer US, pp. 580–589. ISBN: 978-0-387-68854-1.

Jones, D. L., K. Wagstaff, et al. (2012). "Big data challenges for large radio arrays". In: *2012 IEEE Aerospace Conference*, pp. 1–6. DOI: 10.1109/AERO.2012.6187090.

Jönsson, Daniel et al. "A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering". In: *Computer Graphics Forum* 33.1, pp. 27–51. DOI: 10.1111/cgf.12252. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12252. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12252.

Józsa, G. I. G. et al. (2007). "Kinematic modelling of disk galaxies. I. A new method to fit tilted rings to data cubes". In: A&A 468, pp. 731–774. DOI: 10.1051/0004-6361:20066164. eprint: astro-ph/0703207.

Jupyter Hub Development Team (2018). *Jupyterhub — jupyterhub 0.9.1 documentation*. URL: https://jupyterhub.readthedocs.io/ (visited on 08/21/2018).

Jurić, M. et al. (2017). "The LSST Data Management System". In: *Astronomical Data Analysis Software and Systems XXV*. Ed. by N. P. F. Lorente, K. Shortridge, and R. Wayth. Vol. 512. Astronomical Society of the Pacific Conference Series, p. 279. arXiv: 1512.07914 [astro-ph.IM].

Kaehler, Ralf et al. (2006). "GPU-Assisted Raycasting for Cosmological Adaptive Mesh Refinement Simulations". In: *Volume Graphics*. Ed. by Raghu Machiraju and Torsten Moeller. The Eurographics Association. ISBN: 3-905673-41-X. DOI: 10.2312/VG/VG06/103-110.

Kajiya, James T. (1986). "The Rendering Equation". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: ACM, pp. 143–150. ISBN: 0-89791-196-2. DOI: 10.1145/15922.15902. URL: http://doi.acm.org/10.1145/15922.15902.

Kajiya, James T. and Brian P Von Herzen (1984). "Ray Tracing Volume Densities". In: *SIGGRAPH Comput. Graph.* 18.3, pp. 165–174. ISSN: 0097-8930. DOI: 10.1145/964965.808594. URL: http://doi.acm.org/10.1145/964965.808594.

Karpatne, Anuj and Vipin Kumar (2017). "Big Data in Climate: Opportunities and Challenges for Machine Learning". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: ACM, pp. 21–22. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3105810. URL: http://doi.acm.org/10.1145/3097983.3105810.

Kent, Brian R. (2017). "Editorial: Techniques and Methods for Astrophysical Data Visualization". In: *Publications of the Astronomical Society of the Pacific* 129.975, p. 058001. URL: http://stacks.iop.org/1538-3873/129/i=975/a=058001.

Kim, Joonseok and Peter W. Groeneveld (2017). "Big Data, Health Informatics, and the Future of Cardiovascular Medicine". In: *Journal of the American College of Cardiology* 69.7, pp. 899–902. ISSN: 0735-1097. DOI: 10.1016/j.jacc.2017.01.006. eprint: http://www.onlinejacc.org/content/69/7/899.full.pdf. URL: http://www.onlinejacc.org/content/69/7/899.

Klassen, M. et al. (2014). "A General Hybrid Radiation Transport Scheme for Star Formation Simulations on an Adaptive Grid". In: ApJ 797, 4, p. 4. DOI: 10.1088/0004-637X/797/1/4. arXiv: 1410.4259.

Koerner, D. et al. (2014). "Flux-Limited Diffusion for Multiple Scattering in Participating Media". In: *Comput. Graph. Forum* 33.6, pp. 178–189. ISSN: 0167-7055. DOI: 10.1111/cgf.12342. URL: https://doi.org/10.1111/cgf.12342.

Kogge, Peter et al. (2008). *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead*. U.S. Defense Advanced Research Projects Agency.

Koribalski, B. S. et al. (2018). "The Local Volume H I Survey (LVHIS)". In: MNRAS 478, pp. 1611–1648. DOI: 10.1093/mnras/sty479.

Kowalkowski, J (2014). "Data processing in the wake of massive multi-core processors". In: *Journal of Physics: Conference Series* 513.5, p. 052015. URL: http://stacks.iop.org/1742-6596/513/i=5/a=052015.

Kurtzer, Gregory M., Vanessa Sochat, and Michael W. Bauer (2017). "Singularity: Scientific containers for mobility of compute". In: *PLOS ONE* 12.5, pp. 1–20. DOI: 10.1371/journal.pone.0177459. URL: https://doi.org/10.1371/journal.pone.0177459.

Lacroute, Philippe and Marc Levoy (1994). "Fast Volume Rendering Using a Shear-warp Factorization of the Viewing Transformation". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA: ACM, pp. 451–458. ISBN: 0-89791-667-0. DOI: 10.1145/192161.192283. URL: http://doi.acm.org/10.1145/192161.192283.

Laney, Doug (2001). *3D Data Management: Controlling Data Volume, Velocity, and Variety*.

Lang, Ulrich and Michel Grave (1993). "Data Structures in Scientific Visualization". In: *Focus on Scientific Visualization*. Ed. by Hans Hagen, Heinrich Müller, and Gregory M. Nielson. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 85–102. ISBN: 978-3-642-77165-1. DOI: 10.1007/978-3-642-77165-1_4. URL: https://doi.org/10.1007/978-3-642-77165-1_4.

Lawrence, E. et al. (2010). "The Coyote Universe. III. Simulation Suite and Precision Emulator for the Nonlinear Matter Power Spectrum". In: ApJ 713, pp. 1322–1331. DOI: 10.1088/0004-637X/713/2/1322. arXiv: 0912.4490 [astro-ph.CO].

Leaf, N. et al. (2013). "Efficient parallel volume rendering of large-scale adaptive mesh refinement data". In: *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pp. 35–42. DOI: 10.1109/LDAV.2013.6675156.

Lemson, G. and t. Virgo Consortium (2006). "Halo and Galaxy Formation Histories from the Millennium Simulation: Public release of a VO-oriented and SQL-queryable database for studying the evolution of galaxies in the LambdaCDM cosmogony". In: *ArXiv Astrophysics e-prints*. eprint: astro-ph/0608019.

Levermore, C. D. and G. C. Pomraning (1981). "A flux-limited diffusion theory". In: ApJ 248, pp. 321–334. DOI: 10.1086/159157.

Li, H., C. Fu, and A. Hanson (2008). "Visualizing Multiwavelength Astrophysical Data". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6, pp. 1555–1562. ISSN: 1077-2626. DOI: 10.1109/TVCG.2008.182.

Lipsa, Dan R. et al. (2012). "Visualization for the Physical Sciences". In: *Comput. Graph. Forum* 31.8, pp. 2317–2347. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03184.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03184.x.

Liu, B. et al. (2012). "Accelerating High Performance Computing Applications: Using CPUs, GPUs, Hybrid CPU/GPU, and FPGAs". In: *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 337–342. DOI: 10.1109/PDCAT.2012.34.

Ljung, Patric et al. (2016). "State of the Art in Transfer Functions for Direct Volume Rendering". In: *Computer Graphics Forum* 35.3, pp. 669–691. DOI: 10.1111/cgf.12934. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12934. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12934.

Lobeiras, Jacobo, Margarita Amor, and Ramón Doallo (2012). "Influence of memory access patterns to small-scale FFT performance". In: *The Journal of Supercomputing* 64, pp. 120–131.

Lorensen, William E. and Harvey E. Cline (1987). "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: ACM, pp. 163–169. ISBN: 0-89791-227-6. DOI: 10.1145/37401.37422. URL: http://doi.acm.org/10.1145/37401.37422.

M., Jacobsen D. and Canon R. S. (2015). "Contain this, unleashing docker for hpc." In: *Proceedings of the Cray User Group*.

Ma, Kwan-Liu et al. (1994). "Parallel Volume Rendering Using Binary-Swap Compositing". In: *IEEE Comput. Graph. Appl.* 14.4, pp. 59–68. ISSN: 0272-1716. DOI: 10.1109/38.291532. URL: https://doi.org/10.1109/38.291532.

MacCrann, N et al. (2018). "DES Y1 Results: validating cosmological parameter estimation using simulated Dark Energy Surveys". In: *Monthly Notices of the Royal Astronomical Society* 480.4, pp. 4614–4635. DOI: 10.1093/mnras/sty1899. eprint: /oup/backfile/content_public/journal/mnras/480/4/10.1093_mnras_sty1899/1/sty1899.pdf. URL: http://dx.doi.org/10.1093/mnras/sty1899.

Madduri, R. et al. (2015). "PDACS: A Portal for Data Analysis Services for Cosmological Simulations". In: *Computing in Science Engineering* 17.5, pp. 18–26. ISSN: 1521-9615. DOI: 10.1109/MCSE.2015.87.

Magnor, M. A., K. Hildebrand, et al. (2005). "Reflection nebula visualization". In: *VIS 05. IEEE Visualization, 2005.* Pp. 255–262. DOI: 10.1109/VISUAL.2005.1532803.

Magnor, M., G. Kindlmann, et al. (2004). "Constrained inverse volume rendering for planetary nebulae". In: *IEEE Visualization 2004*, pp. 83–90. DOI: 10.1109/VISUAL.2004.18.

Magnor, Marcus, Pradeep Sen, et al. (2010). "Progress in Rendering and Modeling for Digital Planetariums". In: *Eurographics 2010 - Areas Papers*. Ed. by Matthew Cooper and Kari Pulli. The Eurographics Association. DOI: 10.2312/ega.20101000.

Malladi, Rama, Richard Dodson, and Vyacheslav Kitaeff (2012). "Intel®Many Integrated Core (MIC) Architecture: Portability and Performance Efficiency Study of Radio Astronomy Algorithms". In: *Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy Date*. Astro-HPC '12. Delft, The Netherlands: ACM, pp. 5–6. ISBN: 978-1-4503-1338-4. DOI: 10.1145/2286976.2286979. URL: http://doi.acm.org/10.1145/2286976.2286979.

Manning, G. (1989). "The use of the DAP, a massively parallel computing system, for information retrieval and processing". In: *IEE Colloquium on Parallel Techniques for Information Retrieval*, pp. 4/1–4/5.

Marco K. Bosma Jaap Smit, Steven Lobregt (1998). "Iso-surface volume rendering". In: vol. 3335, pp. 3335–10. DOI: 10.1117/12.312490. URL: https://doi.org/10.1117/12.312490.

Marinescu, D. C. (2013). *Cloud computing*. Morgan Kaufmann.

Markov, Igor L. (2014). "Limits on fundamental limits to computation". In: *Nature* 512. Review Article. URL: http://dx.doi.org/10.1038/nature13570.

Marmitt, Gerd, Heiko Friedrich, and Philipp Slusallek (2008). "Efficient CPU-based Volume Ray Tracing Techniques". In: *Computer Graphics Forum* 27.6, pp. 1687–1709. DOI: 10.1111/j.1467-8659.2008.01179.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2008.01179.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01179.x.

Mauch, V., M. Kunze, and M. Hillenbrand (2013). "High performance cloud computing". In: *Future Generation Computer Systems* 29.6, pp. 1408–1416. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2012.03.011. URL: http://www.sciencedirect.com/science/article/pii/S0167739X12000647.

Max, Nelson (1995). "Optical Models for Direct Volume Rendering". In: *IEEE Transactions on Visualization and Computer Graphics* 1.2, pp. 99–108. ISSN: 1077-2626. DOI: 10.1109/2945.468400. URL: https://doi.org/10.1109/2945.468400.

Max, Nelson L. and Min Chen (2010). "Local and Global Illumination in the Volume Rendering Integral". In: *Scientific Visualization: Advanced Concepts*, pp. 259–274.

McAlpine, S. et al. (2016). "The EAGLE simulations of galaxy formation: Public release of halo and galaxy catalogues". In: *Astronomy and Computing* 15, pp. 72–89. DOI: 10.1016/j.ascom.2016.02.004. arXiv: 1510.01320.

McCormick, B., T. DeFanti, and M. Brown, eds. (1987). Vol. 21. 6. New York, NY, USA: ACM.

McCue, Molly E. and Annette M. McCoy (2017). "The Scope of Big Data in One Medicine: Unprecedented Opportunities and Challenges". In: *Frontiers in Veterinary Science* 4, p. 194. ISSN: 2297-1769. DOI: 10.3389/fvets.2017.00194. URL: https://www.frontiersin.org/article/10.3389/fvets.2017.00194.

Meissner, Michael et al. (2000). "A Practical Evaluation of Popular Volume Rendering Algorithms". In: *Proceedings of the 2000 IEEE Symposium on Volume Visualization*. VVS '00. Salt Lake City, Utah, USA: ACM, pp. 81–90. ISBN: 1-58113-308-1. DOI: 10.1145/353888.353903. URL: http://doi.acm.org/10.1145/353888.353903.

Merloni, A. et al. (2012). "eROSITA Science Book: Mapping the Structure of the Energetic Universe". In: *ArXiv e-prints*. arXiv: 1209.3114 [astro-ph.HE].

Meuer, Hans et al. (1999). *Top500 List November 1999*. URL: https://www.top500.org/lists/1993/11/ (visited on 08/17/2018).

— (2006). *Top500 List November 2006*. URL: https://www.top500.org/list/2006/11/ (visited on 08/17/2018).

Meurer, G. R. et al. (2006). "The Survey for Ionization in Neutral Gas Galaxies. I. Description and Initial Results". In: ApJS 165, pp. 307–337. DOI: 10.1086/504685. eprint: astro-ph/0604444.

Miller, Michael (2009). *Cloud computing*. Que.

Milligan, Michael (2017). "Interactive HPC Gateways with Jupyter and Jupyterhub". In: *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*. PEARC17. New Orleans, LA, USA: ACM, 63:1–63:4. ISBN: 978-1-4503-5272-7. DOI: 10.1145/3093338.3104159. URL: http://doi.acm.org/10.1145/3093338.3104159.

Mitra, T. and T. -. Chiueh (1998). "Implementation and evaluation of the parallel Mesa library". In: *Proceedings 1998 International Conference on Parallel and Distributed Systems (Cat. No.98TB100250)*, pp. 84–91. DOI: 10.1109/ICPADS.1998.741023.

Monaghan, J. J. and J. C. Lattanzio (1985). "A refined particle method for astrophysical problems". In: A&A 149, pp. 135–143.

Monaghan, J.J. (1988). "An introduction to SPH". In: *Computer Physics Communications* 48.1, pp. 89–96. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(88)90026-4. URL: http://www.sciencedirect.com/science/article/pii/0010465588900264.

Moore, Gordon E. (1965). "Cramming More Components onto Integrated Circuits". In: *Electronics* 38.8, pp. 114–117.

Moreland, Kenneth et al. (2011). "An Image Compositing Solution at Scale". In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '11. Seattle, Washington: ACM, 25:1–25:10. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063417. URL: http://doi.acm.org/10.1145/2063384.2063417.

Morgan, K and J Peraire (1998). "Unstructured grid finite-element methods for fluid mechanics". In: *Reports on Progress in Physics* 61.6, p. 569. URL: http://stacks.iop.org/0034-4885/61/i=6/a=001.

Mucci, Philip J. et al. (1999). "PAPI: A Portable Interface to Hardware Performance Counters". In: *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pp. 7–10.

Mueller, K. and R. Crawfis (1998). "Eliminating popping artifacts in sheet buffer-based splatting". In: *Proceedings Visualization '98 (Cat. No.98CB36276)*, pp. 239–245. DOI: 10.1109/VISUAL.1998.745309.

Murray, S. D. et al. (1994). "Accretion disk coronae in high-luminosity systems". In: ApJ 435, pp. 631–646. DOI: 10.1086/174842. eprint: astro-ph/9405016.

Mwalongo, F. et al. (2016). "State-of-the-Art Report in Web-based Visualization". In: *Comput. Graph. Forum* 35.3, pp. 553–575. ISSN: 0167-7055. DOI: 10.1111/cgf.12929. URL: https://doi.org/10.1111/cgf.12929.

Nadeau, David R. et al. (2001). "Visualizing Stars and Emission Nebulas". In: *Computer Graphics Forum*. ISSN: 1467-8659. DOI: 10.1111/1467-8659.00472.

Nelson, D. et al. (2015). "The illustris simulation: Public data release". In: *Astronomy and Computing* 13, pp. 12–37. DOI: 10.1016/j.ascom.2015.09.003. arXiv: 1504.00362.

Norris, R. P. (2011). "Data Challenges for Next-generation Radio Telescopes". In: *Sixth IEEE International Conference on eScience, p. 21-24*, pp. 21–24. DOI: 10.1109/eScienceW.2010.13. arXiv: 1101.1355 [astro-ph.IM].

Nulkar, Manjushree and Klaus Mueller (2001). "Splatting With Shadows". In: *Volume Graphics 2001*. Ed. by Klaus Mueller and Arie E. Kaufman. Vienna: Springer Vienna, pp. 35–49. ISBN: 978-3-7091-6756-4.

Nunes, D. S., P. Zhang, and J. Sá Silva (2015). "A Survey on Human-in-the-Loop Applications Towards an Internet of All". In: *IEEE Communications Surveys Tutorials* 17.2, pp. 944–965. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2398816.

NVIDIA Corporation (2017a). *NVIDIA TESLA V100 GPU Architecture*. Tech. rep.

NVIDIA Corporation (2017b). *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*. Tech. rep.

— (2018a). *Nsight Eclipse Edition CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/nsight-eclipse-edition-getting-started-guide/index.html (visited on 08/22/2018).

— (2018b). *NVIDIA CUDA C Programming Guide*. NVIDIA Corporation.

— (2018c). *NVIDIA CUDA Toolkit Profiler User's Guide*. URL: https://docs.nvidia.com/cuda/profiler-users-guide/index.html (visited on 08/22/2018).

Nystrom, Nicholas A. et al. (2015). "Bridges: A Uniquely Flexible HPC Resource for New Communities and Data Analytics". In: *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. XSEDE '15. St. Louis, Missouri: ACM, 30:1–30:8. ISBN: 978-1-4503-3720-5. DOI: 10.1145/2792745.2792775. URL: http://doi.acm.org/10.1145/2792745.2792775.

O'Brien, J. C., K. C. Freeman, and P. C. van der Kruit (2010). "The dark matter halo shape of edge-on disk galaxies. IV. UGC 7321". In: A&A 515, A63, A63. DOI: 10.1051/0004-6361/200912568. arXiv: 1002.3098 [astro-ph.CO].

Ochsenbein, F., P. Bauer, and J. Marcout (2000). "The VizieR database of astronomical catalogues". In: A&AS 143, pp. 23–32. DOI: 10.1051/aas:2000169. eprint: astro-ph/0002122.

Overzier, R. et al. (2013). "The Millennium Run Observatory: first light". In: MNRAS 428, pp. 778–803. DOI: 10.1093/mnras/sts076. arXiv: 1206.6923.

Pascucci, G. et al. (2012). "The ViSUS Visualization Framework". In: *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Ed. by E. Wes Bethel, Hank Childs, and Charles Hansen. Florida: Chapman & Hall/CRC. Chap. 19.

Patidar, S., D. Rane, and P. Jain (2012). "A Survey Paper on Cloud Computing". In: *2012 Second International Conference on Advanced Computing Communication Technologies*, pp. 394–398. DOI: 10.1109/ACCT.2012.15.

Pawlik, Andreas H. and Joop Schaye (2008). "traphic– radiative transfer for smoothed particle hydrodynamics simulations". In: *Monthly Notices of the Royal Astronomical Society* 389.2, pp. 651–677. DOI: 10.1111/j.1365-2966.2008.13601.x. eprint: /oup/backfile/content_public/journal/mnras/389/2/10.1111/j.1365-2966.2008.13601.x/2/mnras0389-0651.pdf. URL: http://dx.doi.org/10.1111/j.1365-2966.2008.13601.x.

Pence, W. D. et al. (2010). "Definition of the Flexible Image Transport System (FITS), version 3.0". In: A&A 524, A42, A42. DOI: 10.1051/0004-6361/201015362.

Penny, S. J. et al. (2015). "Galaxy And Mass Assembly (GAMA): the bright void galaxy population in the optical and mid-IR". In: MNRAS 453, pp. 3519–3539. DOI: 10.1093/mnras/stv1926. arXiv: 1508.06186.

Pennycook, S. J., J. D. Sewall, and V. W. Lee (2016). "A Metric for Performance Portability". In: *arXiv e-prints*, arXiv:1611.07409, arXiv:1611.07409. arXiv: 1611.07409 [cs.PF].

Perlin, Ken (1985). "An Image Synthesizer". In: *SIGGRAPH Comput. Graph.* 19.3, pp. 287–296. ISSN: 0097-8930. DOI: 10.1145/325165.325247. URL: http://doi.acm.org/10.1145/325165.325247.

Peskin, R. L. et al. (1991). "Interactive Quantitative Visualization". In: *IBM Journal of Research and Development* 35.1.2, pp. 205–226. ISSN: 0018-8646. DOI: 10.1147/rd.351.0205.

Peterka, T. et al. (2009). "A configurable algorithm for parallel image-compositing applications". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–10. DOI: 10.1145/1654059.1654064.

Peters, Thomas (2014). "The physics of volume rendering". In: *European Journal of Physics* 35, 065028, p. 065028. DOI: 10.1088/0143-0807/35/6/065028. arXiv: 1410.6022 [astro-ph.IM].

Pfister, H. et al. (2001). "The transfer function bake-off". In: *IEEE Computer Graphics and Applications* 21.3, pp. 16–22. ISSN: 0272-1716. DOI: 10.1109/38.920623.

Potter, Douglas, Joachim Stadel, and Romain Teyssier (2017). "PKDGRAV3: beyond trillion particle cosmological simulations for the next era of galaxy surveys". In: *Computational Astrophysics and Cosmology* 4.1, p. 2. DOI: 10.1186/s40668-017-0021-1. URL: https://doi.org/10.1186/s40668-017-0021-1.

Price, D. J. (2007). "splash: An Interactive Visualisation Tool for Smoothed Particle Hydrodynamics Simulations". In: PASA 24, pp. 159–173. DOI: 10.1071/AS07022. arXiv: 0709.0832.

Ragagnin, A. et al. (2016). "An online theoretical virtual observatory for hydrodynamical, cosmological simulations". In: *ArXiv e-prints*. arXiv: 1612.06380 [astro-ph.IM].

Raji, M., A. Hota, and J. Huang (2017). "Scalable web-embedded volume rendering". In: *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 45–54. DOI: 10.1109/LDAV.2017.8231850.

Rajovic, Nikola et al. (2013). "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 40:1–40:12. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503281. URL: http://doi.acm.org/10.1145/2503210.2503281.

Ralf Kaehler, Tom Abel (2013). "Single-pass GPU-raycasting for structured adaptive mesh refinement data". In: vol. 8654, pp. 8654–12. DOI: 10.1117/12.2008552. URL: https://doi.org/10.1117/12.2008552.

Reed, Daniel A. and Jack Dongarra (2015). "Exascale Computing and Big Data". In: *Commun. ACM* 58.7, pp. 56–68. ISSN: 0001-0782. DOI: 10.1145/2699414. URL: http://doi.acm.org/10.1145/2699414.

Reed, Darren, Tim Dykes, et al. (2018). "DIAPHANE: a portable radiation transport library for astrophysical applications". English. In: *Computer Physics Communications* 226, pp. 1–9. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2017.11.009.

Reid, F. and I. Bethune (2014). "Optimising CP2K for the Intel Xeon Phi". In: *PRACE White Paper*, http://www.prace-ri.eu/evaluation-intel-mic.

Rimal, B. P., E. Choi, and I. Lumb (2009). "A Taxonomy and Survey of Cloud Computing Systems". In: *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44–51. DOI: 10.1109/NCM.2009.218.

Ritzerveld, Jelle, Vincent Icke, and Erik-Jan Rijkhorst (2003). "Triangulating Radiation: Radiative Transfer on Unstructured Grids". In: *arXiv e-prints*, astro-ph/0312301, astro–ph/0312301. arXiv: astro-ph/0312301 [astro-ph].

Rivi, M. et al. (2014). "GPU accelerated particle visualisation with Splotch". In: *Astronomy and Computing* 5. 12 months embargo., pp. 9–18. ISSN: 2213-1337. DOI: 10.1016/j.ascom.2014.03.001.

Rogers, P. D. and J. Wadsley (2011). "The importance of photosphere cooling in simulations of gravitational instability in the inner regions of protostellar discs". In: *Monthly Notices of the Royal Astronomical Society* 414, pp. 913–929. DOI: 10.1111/j.1365-2966.2011.18523.x.

Rogstad, D. H., I. A. Lockhart, and M. C. H. Wright (1974). "Aperture-synthesis observations of H I in the galaxy M83." In: ApJ 193, pp. 309–319. DOI: 10.1086/153164.

Rushmeier, Holly (1995). "Rendering Participating Media: Problems and Solutions from Application Areas". In: *Photorealistic Rendering Techniques*. Ed. by Georgios Sakas, Stefan Müller, and Peter Shirley. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 37–59. ISBN: 978-3-642-87825-1.

Rybicki, G. B. and A. P. Lightman (1979). *Radiative processes in astrophysics*.

Schroeder, W., K. Martin, and B. Lorensen (2006). *The Visualization Toolkit*. Vol. 4. Kitware. ISBN: 978-1-930934-19-1.

Sciacca, E., M. Bandieramonte, et al. (2013). "VisIVO Workflow-Oriented Science Gateway for Astrophysical Visualization". In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 164–171. DOI: 10.1109/PDP.2013.31.

Sciacca, Eva, Ugo Becciani, et al. (2015). "An integrated visualization environment for the virtual observatory: current status and future directions". In: *Astronomy and Computing* 11.Part B. 12 month embargo., pp. 146–154. ISSN: 2213-1337. DOI: 10.1016/j.ascom.2015.01.006.

Searcy, Craig, Ken Dean, and William Stringer (1998). "PUFF: A high-resolution volcanic ash tracking model". In: *Journal of Volcanology and Geothermal Research* 80.1, pp. 1–16. ISSN: 0377-0273. DOI: https://doi.org/10.1016/S0377-0273(97)00037-1. URL: http://www.sciencedirect.com/science/article/pii/S0377027397000371.

Sivarajah, Uthayasankar et al. (2017). "Critical analysis of Big Data challenges and analytical methods". In: *Journal of Business Research* 70, pp. 263–286. ISSN: 0148-2963. DOI: https://doi.org/10.1016/j.jbusres.2016.08.001. URL: http://www.sciencedirect.com/science/article/pii/S014829631630488X.

Springel, V. (2005). "The cosmological simulation code GADGET-2". In: MNRAS 364, pp. 1105–1134. DOI: 10.1111/j.1365-2966.2005.09655.x. eprint: astro-ph/0505010.

Springel, Volker (2016). "High Performance Computing and Numerical Modelling". In: *Star Formation in Galaxy Evolution: Connecting Numerical Models to Reality: Saas-Fee Advanced Course 43. Swiss Society for Astrophysics and Astronomy*. Ed. by Yves Revaz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 251–358. ISBN: 978-3-662-47890-5. DOI: 10.1007/978-3-662-47890-5_3. URL: https://doi.org/10.1007/978-3-662-47890-5_3.

Springel, V. et al. (2005). "Simulations of the formation, evolution and clustering of galaxies and quasars". In: Nature 435, pp. 629–636. DOI: 10.1038/nature03597. eprint: astro-ph/0504097.

Steffen, Wolfgang et al. (2011). "Shape: A 3D Modeling Tool for Astrophysics". In: *IEEE Transactions on Visualization and Computer Graphics* 17, pp. 454–465. DOI: 10.1109/TVCG.2010.62. arXiv: 1003.2012 [astro-ph.IM].

Strohmaier, Erich et al. (2005). "Recent trends in the marketplace of high performance computing". In: *Parallel Computing* 31.3, pp. 261–273. ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2005.02.001. URL: http://www.sciencedirect.com/science/article/pii/S0167819105000177.

Sutter, Herb and James Larus (2005). "Software and the Concurrency Revolution". In: *Queue* 3.7, pp. 54–62. ISSN: 1542-7730. DOI: 10.1145/1095408.1095421. URL: http://doi.acm.org/10.1145/1095408.1095421.

Szalay, A. (2011). "Extreme Data-Intensive Scientific Computing". In: *Computing in Science Engineering* 13.6, pp. 34–41. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.74.

Szalay, A. and J. Gray (2006). "2020 Computing: Science in an exponential world". In: *Nature* 440, pp. 413–414. DOI: 10.1038/440413a.

Szulágyi, J., L. Mayer, and T. Quinn (2017). "Circumplanetary discs around young giant planets: a comparison between core-accretion and disc instability". In: *Monthly Notices of the Royal Astronomical Society* 464, pp. 3158–3168. DOI: 10.1093/mnras/stw2617. arXiv: 1610.01791 [astro-ph.EP].

Tanenbaum, Andrew S. and Maarten van Steen (2007). *Distributed Systems: Principles and Paradigms*. 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall. ISBN: 978-0-13-239227-3.

Tate, Adrian et al. (2014). *Programming Abstractions for Data Locality*. Research Report. PADAL Workshop 2014, April 28–29, Swiss National Supercomputing Center (CSCS), Lugano, Switzerland, p. 54. URL: https://hal.inria.fr/hal-01083080.

Teyssier, R. (2002). "Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES". In: A&A 385, pp. 337–364. DOI: 10.1051/0004-6361:20011817. eprint: astro-ph/0111367.

Thilker, David A. et al. (2005). "Recent Star Formation in the Extreme Outer Disk of M83". In: ApJ 619, pp. L79–L82. DOI: 10.1086/425251. arXiv: astro-ph/0411306 [astro-ph].

Thorne, Kip S (2014). *The science of Interstellar*. 1st ed. W. W. Norton & Company.

Vogelsberger, M. et al. (2014). "Introducing the Illustris Project: simulating the coevolution of dark and visible matter in the Universe". In: MNRAS 444, pp. 1518–1547. DOI: 10.1093/mnras/stu1536. arXiv: 1405.2921.

Vohl, D., D. G. Barnes, et al. (2016). "Large-scale comparative visualisation of sets of multidimensional data". In: *ArXiv e-prints*. arXiv: 1610.00760 [cs.HC].

Vohl, D., C. J. Fluke, et al. (2017). "Real-time colouring and filtering with graphics shaders". In: *Monthly Notices of the Royal Astronomical Society* 471.3, pp. 3323–3346. DOI: 10.1093/mnras/stx1676. eprint: /oup/backfile/content_public/journal/mnras/471/3/10.1093_mnras_stx1676/2/stx1676.pdf. URL: http://dx.doi.org/10.1093/mnras/stx1676.

Wadsley, J. W., J. Stadel, and T. Quinn (2004). "Gasoline: a flexible, parallel implementation of TreeSPH". In: *New Astronomy* 9.2, pp. 137–158. ISSN: 1384-1076. DOI: https://doi.org/10.1016/j.newast.2003.08.004. URL: http://www.sciencedirect.com/science/article/pii/S138410760300143X.

Wald, I. et al. (2017). "OSPRay - A CPU Ray Tracing Framework for Scientific Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1, pp. 931–940. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2599041.

Wang, L., J. Tao, et al. (2008). "Scientific Cloud Computing: Early Definition and Experience". In: *2008 10th IEEE International Conference on High Performance Computing and Communications*, pp. 825–830. DOI: 10.1109/HPCC.2008.38.

Wang, Qiao, Zong-Yan Cao, et al. (2018). "PHoTo N s–A parallel heterogeneous and threads oriented code for cosmological N -body simulation". In: *Research in Astronomy and Astrophysics* 18.6, p. 062. URL: http://stacks.iop.org/1674-4527/18/i=6/a=062.

Wang, Ruonan and Christopher Harris (2013). "Scaling radio astronomy signal correlation on heterogeneous supercomputers using variousdata distribution methodologies". In: *Experimental Astronomy* 36.3, pp. 433–449. ISSN: 1572-9508. DOI: 10.1007/s10686-013-9340-7. URL: https://doi.org/10.1007/s10686-013-9340-7.

Wen, Z. and C. R. O'dell (1995). "A three-dimensional model of the Orion Nebula". In: ApJ 438, pp. 784–793. DOI: 10.1086/175123.

Westover, Lee Alan (1991). "Splatting: A Parallel, Feed-forward Volume Rendering Algorithm". UMI Order No. GAX92-08005. PhD thesis. Chapel Hill, NC, USA.

Yelick, Katherine et al. (2011). *The Magellan Report on Cloud Computing for Science*. U.S. Department of Energy. URL: http://www.nersc.gov/assets/StaffPublications/2012/MagellanFinalReport.pdf (visited on 08/17/2018).

Yoo, Andy B., Morris A. Jette, and Mark Grondona (2003). "SLURM: Simple Linux Utility for Resource Management". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 44–60. ISBN: 978-3-540-39727-4.

Yu, Hongfeng, Chaoli Wang, and Kwan-Liu Ma (2008). "Massively Parallel Volume Rendering Using 2-3 Swap Image Compositing". In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. SC '08. Austin, Texas: IEEE Press, 48:1–48:11. ISBN: 978-1-4244-2835-9. URL: http://dl.acm.org/citation.cfm?id=1413370.1413419.

Zhang, Yanxia and Yongheng Zhao (2015). "Astronomy in the Big Data Era". In: *Data Science Journal* 14, p. 11. DOI: 10.5334/dsj-2015-011.

Zhou, L. and M. Huang (2017). "Challenges of Software Testing for Astronomical Big Data". In: *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 529–532. DOI: 10.1109/BigDataCongress.2017.91.

Zhou, Yichen, Robin M. Weiss, et al. (2013). "WebViz: A Web-Based Collaborative Interactive Visualization System for Large-Scale Data Sets". In: *GPU Solutions to Multi-scale Problems in Science and Engineering*. Ed. by David A. Yuen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 587–606. ISBN: 978-3-642-16405-7. DOI: 10.1007/978-3-642-16405-7_37. URL: https://doi.org/10.1007/978-3-642-16405-7_37.

Zimmermann, H. (1980). "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection". In: *IEEE Transactions on Communications* 28.4, pp. 425–432. ISSN: 0090-6778. DOI: 10.1109/TCOM.1980.1094702.

ZuHone, J. A. et al. (2018). "The Galaxy Cluster Merger Catalog: An Online Repository of Mock Observations from Simulated Galaxy Cluster Mergers". In: *The Astrophysical Journal Supplement Series* 234.1, p. 4. DOI: 10.3847/1538-4365/aa99db. URL: https://doi.org/10.3847%2F1538-4365%2Faa99db.

Zwicker, Matthias et al. (2001). "EWA Volume Splatting". In: *Proceedings of the Conference on Visualization '01*. VIS '01. San Diego, California: IEEE Computer Society, pp. 29–36. ISBN: 0-7803-7200-X. URL: http://dl.acm.org/citation.cfm?id=601671.601674.